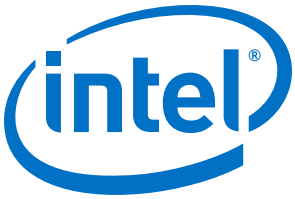


Intel® Quartus® Prime Pro Edition Handbook Volume 1: Design and Compilation

QPP5V1
2017.05.08


Subscribe


Send Feedback

Contents

1 Introduction to Intel Quartus® Prime Pro Edition

The Quartus® Prime software provides a complete design environment for FPGA and SoC designs. The user interface supports easy design entry, fast processing, and straightforward device programming. The Quartus Prime Pro Edition software enables next generation synthesis, physical optimization, design methodologies, and FPGA architectures.

The Quartus Prime Pro Edition Compiler is optimized for the latest Intel Arria® 10 and Intel Cyclone® 10 devices. The Compiler provides powerful and customizable design processing to achieve the best possible design implementation in silicon. The Quartus Prime software makes it easy for you to focus on your design—not on the design tool. The Quartus Prime Pro Edition software provides unique features not available in other Quartus software products.

Figure 1. Quartus Prime New Feature Support Matrix

Software Features	Quartus Prime Standard Edition	Quartus Prime Pro Edition
New Hybrid Placer & Global Router	✓	✓
New TimeQuest	✓	✓
New Physical Synthesis	✓	✓
Incremental Fitter Optimization		✓
BluePrint Platform Designer		✓
New Synthesis Engine		✓
Rapid Recompile		✓
OpenCL		✓
Qsys Pro		✓
Partial Reconfiguration		✓
Block-Based Design Flows		✓

The modular Compiler streamlines the FPGA development process, and ensures the highest performance for the least effort. The Quartus Prime Pro Edition software provides the following unique features:

- Quartus Prime Pro Edition synthesis—integrates new, stricter language parser supporting all major IEEE RTL languages, with enhanced algorithms, and parallel synthesis capabilities. Added support for SystemVerilog 2009.
- Hierarchical project structure—preserves individual post-synthesis, post-placement, and post-place and route results for each design entity. Allows optimization without impacting other partition placement or routing.
- Incremental Fitter Optimizations—run and optimize Fitter stages incrementally. Each Fitter stage generates detailed reports.
- Faster, more accurate I/O placement—plan interface I/O in BluePrint Platform Designer.
- Qsys Pro—builds on the system design and custom IP integration capabilities of Qsys. Qsys Pro introduces hierarchical isolation between system interconnect and IP components.
- Partial Reconfiguration—support reconfiguration of a portion of the Arria 10 FPGA, while the remaining FPGA continues to function.
- Supports block-based design flows, allowing you to preserve and reuse design blocks at various stages of compilation.

Related Links

- Migrating to Quartus Prime Pro Edition on page 17
- Upgrade Project Assignments and Constraints on page 18
- Upgrade IP Cores and Qsys Pro Systems on page 23
- Upgrade Non-Compliant Design RTL on page 24
- Block-Based Design Flows

1.1 Should I Choose the Quartus Prime Pro Edition Software?

Depending on your immediate needs, the Quartus Prime Pro Edition software may be an appropriate choice for your design.

The Quartus Prime Pro Edition software includes many unique features that the Quartus Prime Standard Edition software does not include. However, the Quartus Prime Pro Edition software does not support all features of the Quartus Prime Standard Edition software.

Selecting a Quartus Prime Edition

Consider the requirements and timeline of your project in determining whether the Quartus Prime Standard Edition or Quartus Prime Pro Edition software is most appropriate for you. Use the following factors to inform your decision:

- The Quartus Prime Pro Edition software supports only Arria 10 and Cyclone 10 devices. If your design targets any other Intel FPGA device, select the Quartus Prime Standard Edition.
- Select the Quartus Prime Pro Edition if you are beginning a new Arria 10, Cyclone 10, or Stratix® 10 design, or if your design requires any unique Quartus Prime Pro Edition features, such as Partial Reconfiguration, Qsys Pro, Incremental Optimization, OpenCL support, or Signal Tap routing preservation.
- Quartus Prime Pro Edition software does not support the following Quartus Prime Standard Edition features:
  - I/O Timing Analysis
  - NativeLink third party tool integration
  - Video and Image Processing Suite IP Cores
  - Talkback features
  - Various register merging and duplication settings
  - Saving a node-level netlist as `.vqm`
  - Compare project revisions

Related Links

- Managing Projects
- Design Compilation
- Creating a Partial Reconfiguration Design

1.2 Migrating to Quartus Prime Pro Edition

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

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

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

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

This document describes each migration step in detail.

1.2.1 Keep Pro Edition Project Files Separate

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

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

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

1.2.2 Upgrade Project Assignments and Constraints

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

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

Related Links

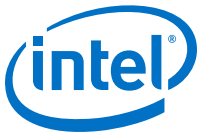
- Modify Entity Name Assignments on page 18
- Resolve SDC Entity Names on page 19
- Verify Generated Node Name Assignments on page 19
- Replace LogicLock Regions on page 20
- Modify Signal Tap Logic Analyzer Files on page 22
- Remove Unsupported Feature Assignments on page 23

1.2.2.1 Modify Entity Name Assignments

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

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

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



1.2.2.2 Resolve SDC Entity Names

The Quartus Prime Pro Edition TimeQuest timing analyzer honors entity names in Synopsys Design Constraints (`.sdc`) files.

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

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

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

# This does the same thing, but first it converts all register names to
# strings, which includes entity names by default in the SE
# but excludes them by default in the PE. The regexp will therefore
# fail in PE by default.
#
# This script would also fail in the SE, and earlier
# versions of Quartus II, if entity name display had been disabled
# in the QSF.
set all_reg_strs [query_collection –list –all [get_registers *]]
foreach keeper $all_reg_strs {
   if {[regexp {counter:*|:*acc} $keeper]} {
      set_false_path –to $keeper
   }
}
```

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

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

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

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

1.2.2.3 Verify Generated Node Name Assignments

Quartus Prime synthesis generates and automatically names internal design nodes during processing. The Quartus Prime Pro Edition uses different conventions than other Quartus software products[1] to generate node names during synthesis. When you synthesize your other Quartus software product project in Quartus Prime Pro Edition,

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

the synthesis-generated node names may change. If any scripts or constraints depend on the synthesis-generated node names, update the script or constraint to match the Quartus Prime Pro Edition synthesis node names.

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

1.2.2.4 Replace LogicLock Regions

Quartus Prime Pro Edition software introduces more simplified and flexible LogicLock® constraints, compared with previous LogicLock regions. You must replace all LogicLock assignments with compatible LogicLock Plus assignments for migration.

To convert LogicLock regions to LogicLock Plus regions:

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

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

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

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

Compilation fails if synthesis finds other Quartus software product LogicLock assignments in a Quartus Prime Pro Edition project. The following table compares other Quartus software product region constraint support with the Quartus Prime Pro Edition software.

Table 1. Region Constraints Per Edition

Constraint Type	LogicLock Region Support Other Quartus Software Products	LogicLock Plus Support Quartus Prime Pro Edition
Fixed rectangular, nonrectangular or non-contiguous regions	Full support.	Full support.
Chip Planner entry	Full support.	Full support.
Periphery element assignments	Supported in some instances.	Full support. Use "core-only" regions to exclude the periphery.
Nested ("hierarchical") regions	Supported but separate hierarchy from the user instance tree.	Supported in same hierarchy as user instance tree.

continued...

Constraint Type	LogicLock Region Support Other Quartus Software Products	LogicLock Plus Support Quartus Prime Pro Edition
Reserved regions	Limited support for nested or nonrectangular reserved regions. Reserved regions typically cannot cross I/O columns; non-contiguous regions must be used instead.	Full support for nested and nonrectangular regions. Reserved regions can cross I/O columns without affecting periphery logic if they are "core-only".
Routing regions	Limited support via "routing expansion." No support with hierarchical regions.	Full support (including future support for hierarchical regions).
Floating or autosized regions	Full support.	No support.
Region names	Regions have names.	Regions are identified by the instance name of the constrained logic.
Multiple instances in the same region	Full support.	Support for non-reserved regions. Create one region per instance, and then specify the same definition for multiple instances to assign to the same area. Not supported for reserved regions.
Member exclusion	Full support.	No support for arbitrary logic. Use a core-only region to exclude periphery elements. Use non-rectangular regions to include more RAM or DSP columns as needed.

1.2.2.4.1 LogicLock Plus Region Assignment Examples

These examples show the syntax for various LogicLock Plus region assignments in the `.qsf` file. Optionally enter these assignments in the Assignment Editor, the LogicLock Regions Window, or the Chip Planner.

Example 1. Assign Rectangular LogicLock Plus Region

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

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

Example 2. Assign Non-Rectangular LogicLock Plus Region

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

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

Example 3. Assign Subordinate LogicLock Plus Instances

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

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

Example 4. Assign Multiple LogicLock Plus Instances

By default, a LogicLock Plus region constraint allows logic from other instances to share the same region. In other words, the following assignments are not in conflict. This assignment places instance c and instance g together. This may be useful if instance c and instance g are heavily interacting.

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

Example 5. Assigned Reserved LogicLock Plus Regions

Optionally reserve an entire LogicLock Plus region for one instance and any of its subordinate instances.

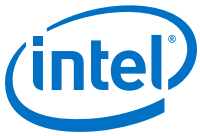
```
set_instance_assignment –name PLACE_REGION –to a|b|c "X10 Y10 X20 Y20"
set_instance_assignment –name RESERVE_PLACE_REGION –to a|b|c ON

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

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

1.2.2.5 Modify Signal Tap Logic Analyzer Files

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

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

1. Remove entity names from `.stp` files. The Signal Tap Logic Analyzer allows without error, but ignores, entity names in `.stp` files. Remove entity names from `.stp` files for migration to Quartus Prime Pro Edition:
   a. Click **View ➤ Node Finder** to locate and remove appropriate nodes. Use Node Finder options to filter on nodes.



b. Click **Processing ➤ Start ➤ Start Analysis & Elaboration** to repopulate the database and add valid node names.

2. Remove post-fit nodes. Quartus Prime Pro Edition uses a different post-fit node naming scheme than other Quartus software products.
   a. Remove post-fit tap node names originating from other Quartus software products.
   b. Click **View ➤ Node Finder** to locate and remove post-fit nodes. Use Node Finder options to filter on nodes.
   c. Click **Processing ➤ Start Compilation** to repopulate the database and add valid post-fit nodes.
3. Run an initial compilation in Quartus Prime Pro Edition from the GUI. The Compiler automatically removes Signal Tap assignments originating other Quartus software products. Alternatively, from the command-line, run `quartus_stp` once on the project to remove outmoded assignments.

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

1.2.2.6 Remove Unsupported Feature Assignments

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

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

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

1.2.3 Upgrade IP Cores and Qsys Pro Systems

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

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

Table 2. IP Core and Qsys Pro System Differences

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

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

Related Links

- Introduction to Intel FPGA IP Cores
- Upgrading IP Cores
- Managing Quartus Prime Projects on page 31

1.2.4 Upgrade Non-Compliant Design RTL

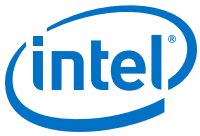
The Quartus Prime Pro Edition software introduces a new synthesis engine (`quartus_syn` executable).

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

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

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

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



Related Links

- Verify Verilog Compilation Unit on page 25
- Update Entity Auto-Discovery on page 26
- Ensure Distinct VHDL Namespace for Each Library on page 26
- Remove Unsupported Parameter Passing on page 27
- Remove Unsized Constant from WYSIWYG Instantiation on page 27
- Remove Non-Standard Pragmas on page 27
- Declare Objects Before Initial Values on page 28
- Confine SystemVerilog Features to SystemVerilog Files on page 28
- Avoid Assignment Mixing in Always Blocks on page 29
- Avoid Unconnected, Non-Existent Ports on page 29
- Avoid Illegal Parameter Ranges on page 29
- Update Verilog HDL and VHDL Type Mapping on page 29

1.2.4.1 Verify Verilog Compilation Unit

Quartus Prime Pro Edition synthesis uses a different method to define the compilation unit. The Verilog LRM defines the concept of compilation unit as "a collection of one or more Verilog source files compiled together" forming the compilation-unit scope. Items visible only in the compilation-unit scope include macros, global declarations, and default net types. The contents of included files become part of the compilation unit of the parent file. Modules, primitives, programs, interfaces, and packages are visible in all compilation units. Ensure that your RTL accommodates these changes.

Table 3. Verilog Compilation Unit Differences

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

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

1.2.4.1.1 Verilog HDL Configuration Instantiation

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

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

Other Quartus Software Products	Quartus Prime Pro Edition
From the Example RTL, synthesis automatically finds the `mid_config` Verilog HDL configuration relating to the instantiated module.	From the Example RTL, synthesis does not find the `mid_config` Verilog HDL configuration. You must instantiate the Verilog HDL configuration directly.

Example RTL:
`config mid_config;`
`design good_lib.mid;`
`instance mid.sub_inst use good_lib.sub;`
`endconfig`
 	
`module test (input a1, output b);`
`mid_config mid_inst ( .a1(a1), .b(b));`
`// in other Quartus products preceding line would have been:`
`//mid mid_inst ( .a1(a1), .b(b));`
`endmodule`
 	
`module mid (input a1, output b);`
`sub sub_inst (.a1(a1), .b(b));`
`endmodule`	

1.2.4.2 Update Entity Auto-Discovery

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

Table 4. Entity Auto-Discovery Differences

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

1.2.4.3 Ensure Distinct VHDL Namespace for Each Library

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

Table 5. VHDL Namespace Differences

Other Quartus Software Products	Quartus Prime Pro Edition
For the Example RTL, the analyzer searches all libraries in an unspecified order until it finds package `utilities_pack` and uses items from that package. If another library, for example `projectLib`	For the Example RTL, the analyzer uses the specific `utilities_pack` in `myLib`. If `utilities_pack` does not exist in library `myLib`, the analyzer generates an error.

continued...

Other Quartus Software Products	Quartus Prime Pro Edition
also contains `utilities_pack`, the analyzer may use this library instead of `myLib.utilites_pack` if found before the analyzer searches `myLib`.	
Example RTL: `library myLib; use myLib.utilities_pack.all;`	

1.2.4.4 Remove Unsupported Parameter Passing

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

Table 6. SystemVerilog Feature Differences

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

1.2.4.5 Remove Unsized Constant from WYSIWYG Instantiation

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

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

1.2.4.6 Remove Non-Standard Pragmas

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

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

1.2.4.7 Declare Objects Before Initial Values

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

Table 7. Object Declaration Differences

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

Example RTL:
`1 output p_prog_io1 = p_prog_io1_reg;`
`2 reg p_prog_io1_reg;`	

1.2.4.8 Confine SystemVerilog Features to SystemVerilog Files

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

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

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

Table 8. SystemVerilog Feature Differences

Other Quartus Software Products	Quartus Prime Pro Edition
From the Example RTL, synthesis interprets `$clog2` in a `.v` file, even though the Verilog LRM does not define the `$clog2` feature. Other Quartus software products allow other SystemVerilog features in `.v` files.	From the Example RTL, synthesis generates a syntax error for detection of any non-Verilog HDL construct in `.v` files. Quartus Prime Pro Edition synthesis honors SystemVerilog features only in `.sv` files.

Example RTL:
`localparam num_mem_locations = 1050;`
`wire mem_addr [$clog2(num_mem_locations)-1 : 0];`	

1.2.4.9 Avoid Assignment Mixing in Always Blocks

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

Table 9. ALWAYS Block Assignment Differences

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

1.2.4.10 Avoid Unconnected, Non-Existent Ports

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

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

Table 10. Unconnected, Non-Existent Port Differences

Other Quartus Software Products[3]	Quartus Prime Pro Edition
Synthesis allows you to instantiate and name unconnected or non-existent ports that do not exist on the module.	Synthesis generates a syntax error for detection of mixed blocking and non-blocking assignments within an ALWAYS block.

1.2.4.11 Avoid Illegal Parameter Ranges

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

1.2.4.12 Update Verilog HDL and VHDL Type Mapping

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

3 For brevity, this section refers to Quartus Prime Standard Edition, Quartus Prime Lite Edition, and Quartus II software collectively as "other Quartus software products."

1.3 Document Revision History

Table 11. Document Revision History

Date	Version	Changes
2017.05.08	17.0.0	• Removed statement about limitations for safe state machines. The Compiler supports safe state machines. State machine inference is enabled by default. • Added reference to Block-Based Design Flows. • Removed procedure on manaul dynamic synthesis report generation. The Compiler automatically generates dynamic synthesis reports when enabled.
2016.10.31	16.1.0	• Implemented Intel rebranding. • Added reference to Partial Reconfiguration support. • Added to list of Quartus Prime Standard Editionfeatures unsupported by Quartus Prime Pro Edition. • Added topic on Safe State Machine encoding. • Described unsupported Quartus Prime Standard Edition physical synthesis options. • Removed deprecated **Per-Stage Compilation (Beta)** Compilation Flow. • Changed title from "Remove Filling Vectors" to "Remove Unsized Constant".
2016.05.03	16.0.0	• Removed software beta status and revised feature set. • Added topic on Safe State Machine encoding. • Added Generating Dynamic Synthesis Reports. • Corrected statement about Verilog Compilation Unit. • Corrected typo in Modify Entity Name Assignments. • Added description of Fitter Plan, Place and Route stages, reporting, and optimization. • Added **Per-Stage Compilation (Beta)** Compilation Flow. • Added Qsys Pro (beta) information. • Added OpenCL and Signal Tap with routing preservation as unique Pro Edition features. • Clarified limitations for multiple LogicLock Plus instances in the same region.
2015.11.02	15.1.0	• First version of document.

Related Links

Altera Documentation Archive

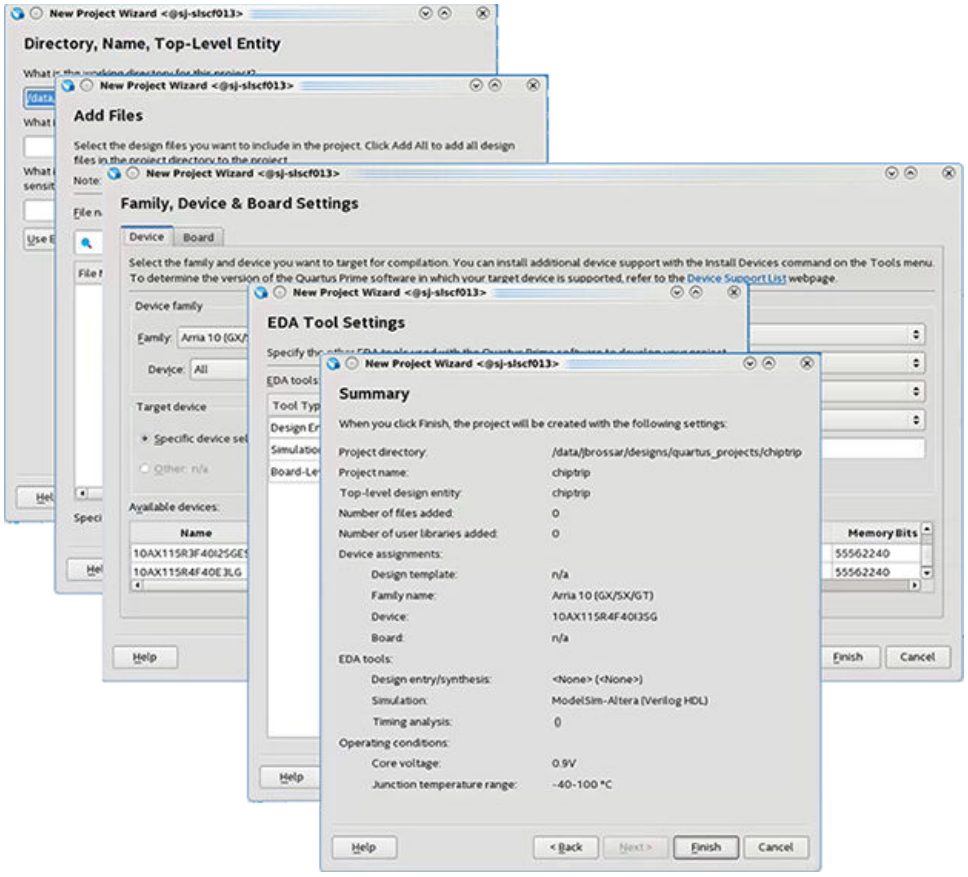
For previous versions of the *Quartus Prime Handbook*, search the Altera documentation archives.

2 Managing Quartus Prime Projects

The Quartus Prime software organizes and manages the elements of your design within a *project*.

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

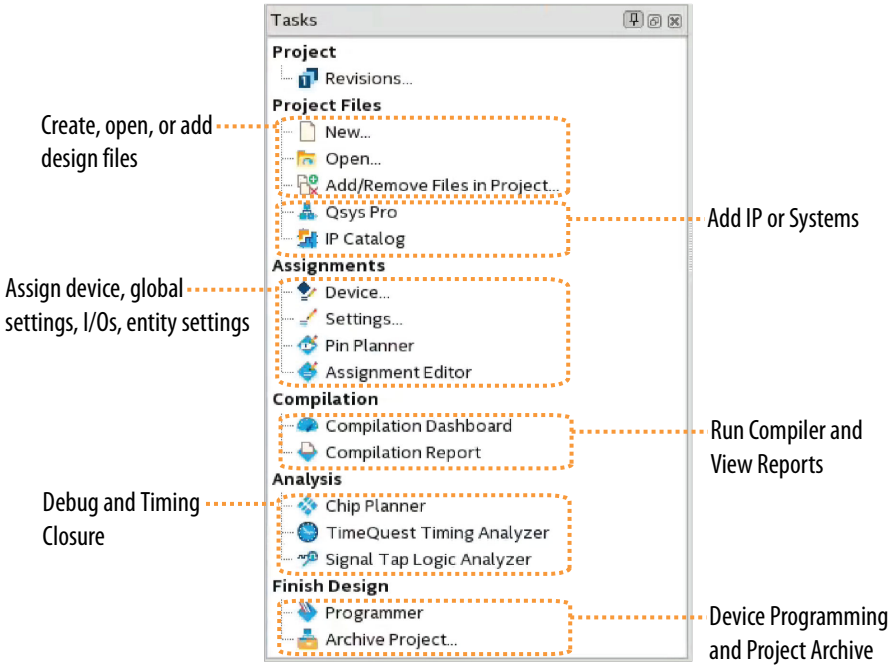
Figure 2. New Project Wizard



When you open a project, a unified GUI displays integrated project information. The project encapsulates information about your design hierarchy, libraries, constraints, and project settings.

Figure 3. Project Tasks Pane

Use the **Tasks** pane for immediate access to all Quartus Prime project settings.



You can save multiple revisions of your project to experiment with settings that achieve your design goals. Quartus Prime projects support team-based, distributed work flows and a scripting interface.

2.1 Understanding Quartus Prime Projects

The Quartus Prime software organizes your FPGA design work within a project. A singleQuartus PrimeProject File (`.qpf`) represents each design project. The text-based **.qpf** references the Quartus Prime Settings File (`.qsf`). The `.qsf` references the project's design, constraint, and IP files, and stores and project-wide or entity-specific settings that you specify in the GUI. The Quartus Prime organizes and maintains these various project files.

Table 12. Quartus Prime Project Files

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

continued...

File Type	Contains	To Edit	Format
Timing constraints	Clock properties, exceptions, setup/hold	**Tools ➤ TimeQuest Timing Analyzer**	Synopsys Design Constraints File (`.sdc`)
Logic design files	RTL and other design source files	**File ➤ New**	All supported HDL files
Programming files	Device programming image and information	**Tools ➤ Programmer**	SRAM Object File (`.sof`)Programmer Object File (`.pof`)
Project library	Project and global library information	**Tools ➤ Options ➤ Libraries**	`.qsf` (project) `quartus2.ini` (global)
IP core files	IP core variation parameterization	**Tools ➤ IP Catalog**	Quartus Prime IP File (`.ip`)
Qsys Pro system files	Qsys Pro system and IP core files	**Tools ➤ Qsys Pro**	Qsys Pro System File (`.qsys`)
EDA tool files	Generated for third-party EDA tools	**Assignments ➤ Settings ➤ EDA Tool Settings**	Verilog Output File (`.vo`) VHDL Output File (`.vho`) Verilog Quartus Mapping File (`.vqm`)
Archive files	Complete project as single compressed file	**Project ➤ Archive Project**	Quartus Prime Archive File (`.qar`)

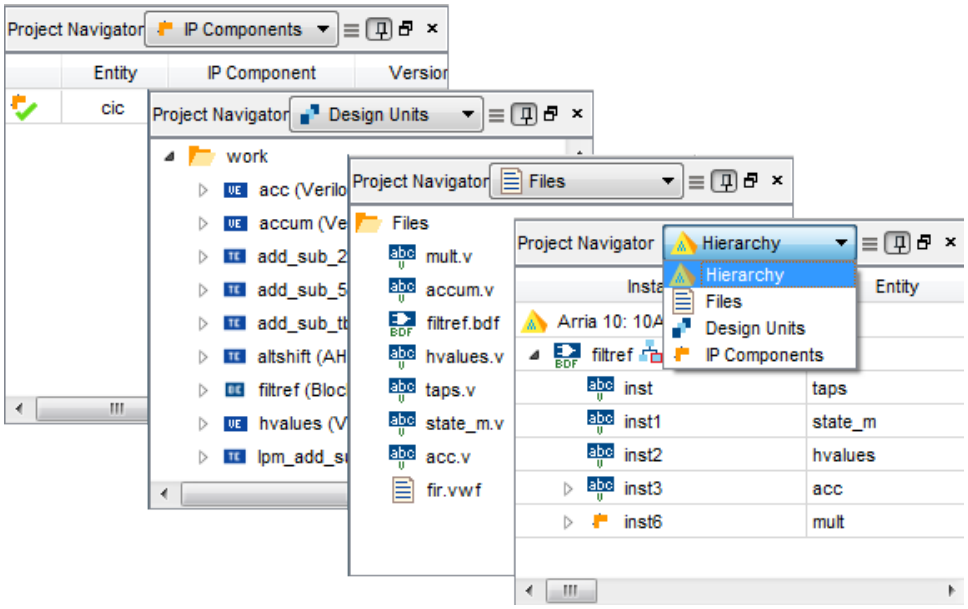
2.2 Viewing Basic Project Information

View basic information about your project in the Project Navigator, Compilation Dashboard, Report panel, and **Messages** window. View project elements in the **Project Navigator** (**View ➤ Project Navigator**). The **Project Navigator** displays key project information, such as design files, IP components, and your project hierarchy. Use the **Project Navigator** to locate and perform actions of the elements of your project. To access the tabs of the Project Navigator, click the toggle control at the top of the **Project Navigator** window.

Table 13. Project Navigator Tabs

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

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



2.2.1 Viewing Project Reports

The Compilation Report panel updates dynamically to display detailed reports during project processing.

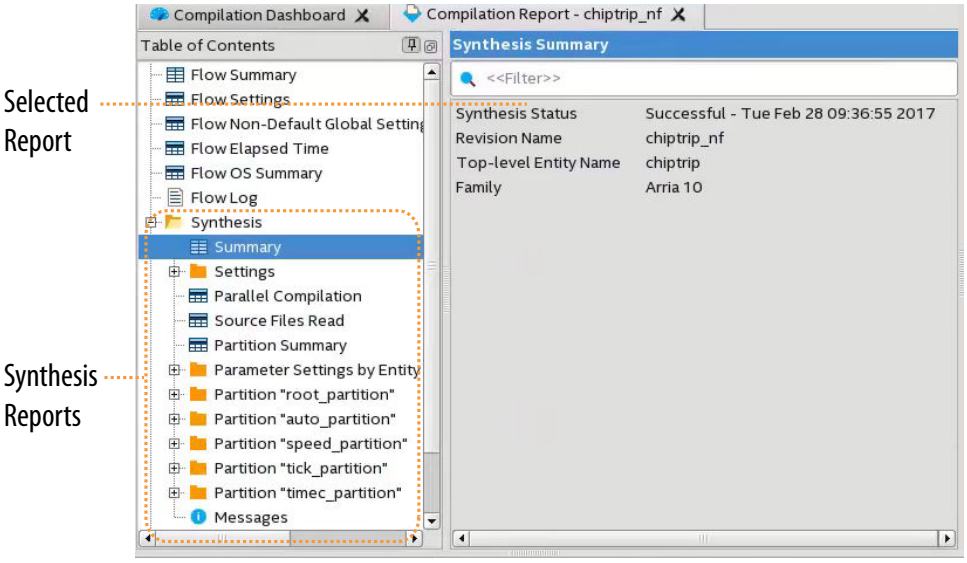
To access the Compilation Report, click (**Processing ➤ Compilation Report**).

Note: You can also access the Compilation Report from the Compilation Dashboard (**Processing ➤ Compilation Dashboard**.

- Synthesis reports
- Fitter reports
- Timing analysis reports
- Power analysis reports
- Signal integrity reports

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

Figure 5. Compilation Report



Related Links

List of Compilation Reports

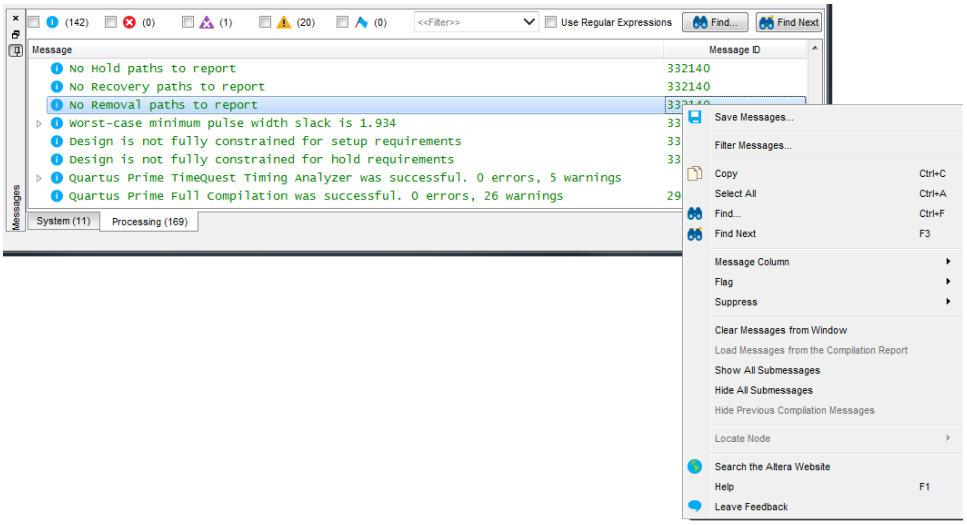
2.2.2 Viewing Project Messages

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

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

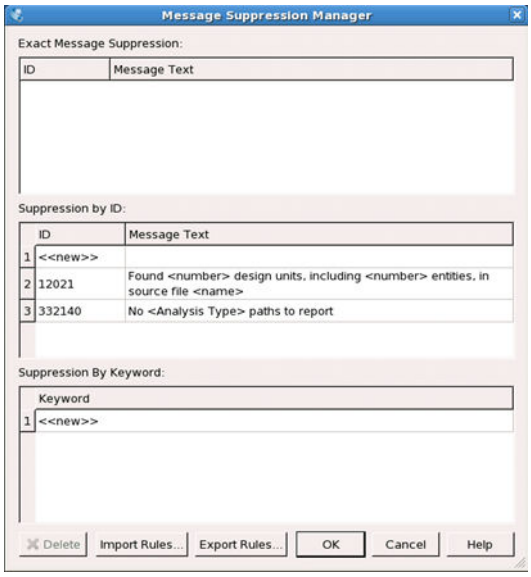
Messages are written to `stdout` when you use command-line executables.

Figure 6. Messages Window



You can suppress display of unimportant messages so they do not obscure valid messages.

Figure 7. Message Suppression by Message ID Number



2.2.2.1 Suppressing Messages

Suppress any messages that you don't want to view. To supress messages, right-click a message and choose any of the following:

- **Suppress Message**—suppresses all messages matching exact text
- **Suppress Messages with Matching ID**—suppresses all messages matching the message ID number, ignoring variables
- **Suppress Messages with Matching Keyword**—suppresses all messages matching keyword or hierarchy

2.2.2.2 Message Suppression Guidelines

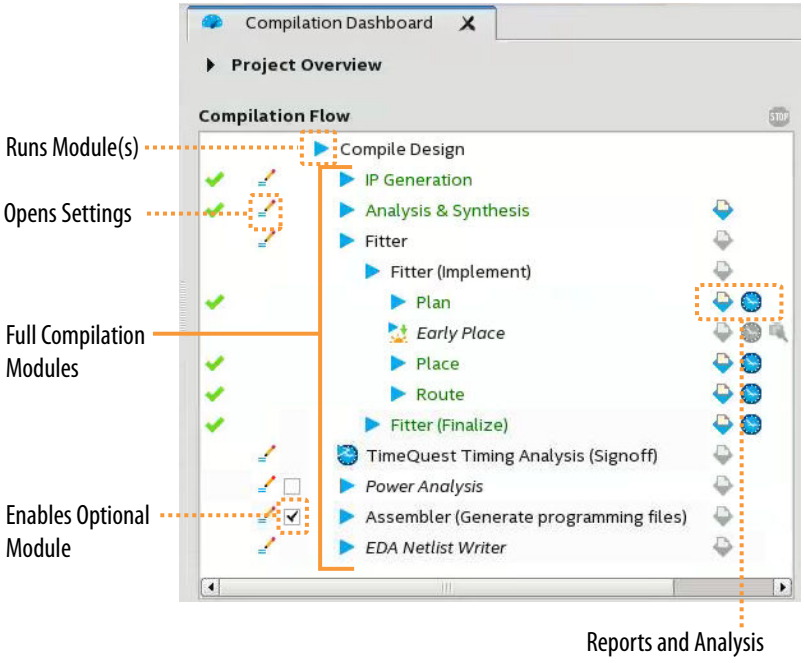
- You cannot suppress error or Intel legal agreement messages.
- Suppressing a message also suppresses any submessages.
- Message suppression is revision-specific. Derivative revisions inherit any suppression.
- You cannot edit messages or suppression rules during compilation.

2.3 Using the Compilation Dashboard

The Compilation Dashboard provides an overview of your project, and lets you change project settings, compile your design, and view reports for each compilation stage.

The Compilation Dashboard appears by default when you open a project. To open the Compilation Dashboard manually, click **Compilation Dashboard** in the Tasks window. You can also access the Compilation Report from the Compilation Dashboard.

Figure 8. Compilation Dashboard



2.4 Project Management Best Practices

The Quartus Prime software provides various options for setting up a project. The following best practices help ensure efficient management and portability of your project files.

Setting and Project File Best Practices

- Be very careful if you edit any Quartus Prime data files, such as the Quartus Prime Project File (`.qpf`), Quartus Prime Settings File (`.qsf`), Quartus IP File (`.qip`), or Qsys Pro System File (`.qsys`). Typos in these files can cause software errors. For example, the software may ignore settings and assignments.

  Every Quartus Prime project revision automatically includes a supporting `.qpf` that preserves various project settings and constraints that you enter in the GUI or add with Tcl commands. This file contains basic information about the current software version, date, and project-wide and entity level settings. Due to dependencies between the `.qpf` and `.qsf`, avoid manually editing `.qsf` files.
- Do not compile multiple projects into the same directory. Instead, use a separate directory for each project.
- By default, the Quartus Prime software saves all project output files, such as Text-Format Report Files (`.rpt`), in the project directory. Instead of manually moving project output files, change your project compilation settings to save them in a separate directory.

  To save these files into a different directory choose **Assignments ➤ Settings ➤ Compilation Process Settings**. Turn on **Save project output files in specified directory** and specify a directory for the output files.

Project Archive and Source Control Best Practices

Click **Project ➤ Archive Project** to archive your project for revision control.

As you develop your design, your Quartus Prime project directory contains a variety of source and settings files, compilation database files, output, and report files. You can archive these files using the Archive feature and save the archive for later use or place it under revision control.

1. Choose **Project ➤ Archive Project ➤ Advanced** to open the **Advanced Archive Settings** dialog box.
2. Choose a file set to archive.
3. Add additional files by clicking **Add** (optional).

To restore your archived project, choose **Project ➤ Restore Archived Project**. Restore your project into a new, empty directory.

IP Core Best Practices

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

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

Related Links

Generating a Combined Simulator Setup Script on page 66

2.5 Managing Project Settings

The New Project Wizard guides you to make intial project settings when you setup a new project. Optimizing project settings helps the Compiler to generate programming files that meet or exceed your specifications.

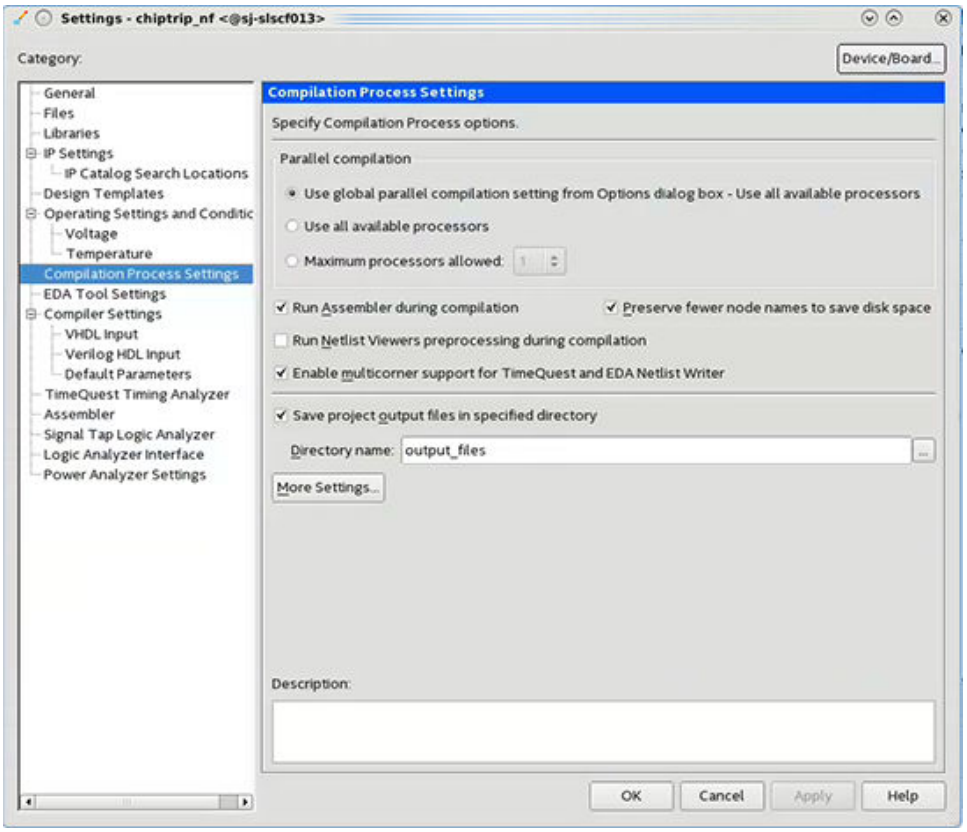
On the **Tasks** pane, click **Settings** to access global project settings, including:

- Project files list
- Synthesis directives and constraints
- Logic options and compiler effort levels
- Placement constraints

- Timing constraint files
- Operating temperature limits and conditions
- File generation for other EDA tools
- Target a device (click **Device**)
- Target a development kit

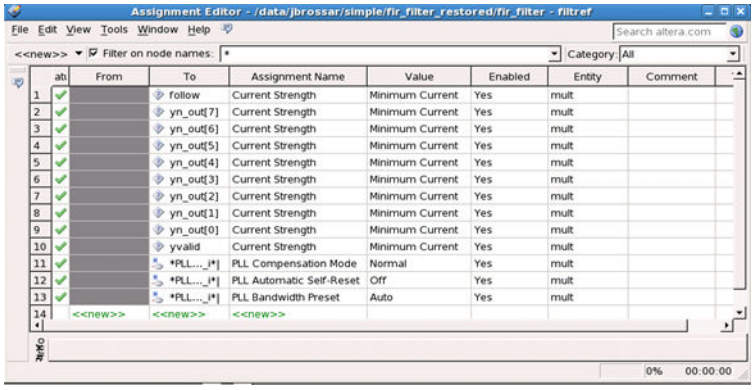
The `.qsf` stores each project revision's project settings. The Quartus Prime Default Settings File (*<revision name>*`_assignment_defaults.qdf`) stores the default settings and constraints for each new project revision.

Figure 9. Settings Dialog Box for Global Project Settings



The Assignment Editor (**Tools > Assignment Editor**) provides a spreadsheet-like interface for assigning all instance-specific settings and constraints.

Figure 10. Assignment Editor Spreadsheet



2.5.1 Optimizing Project Settings

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

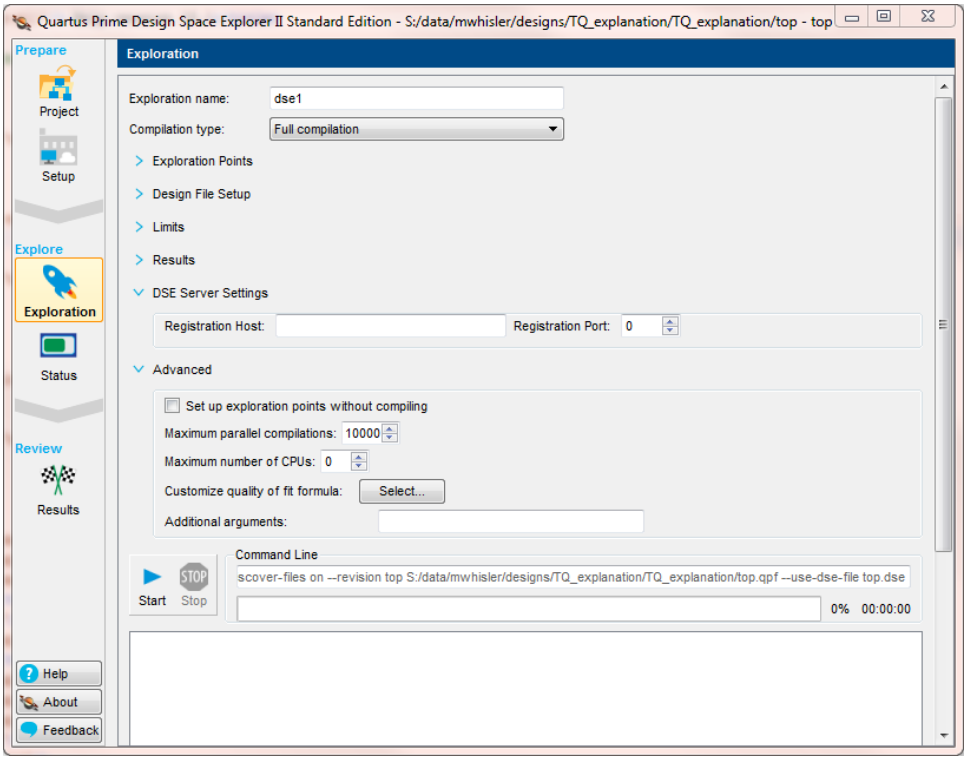
The Quartus Prime software includes several advisors to help you optimize your design and reduce compilation time. The advisors listed in the **Tools ➤ Advisors** menu can provide recommendations based on your project settings and design constraints.

2.5.1.1 Optimizing with Design Space Explorer II

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

DSE II attempts multiple seeds to identify one meeting your requirements. DSE II can run different compilations on multiple computers in parallel to streamline timing closure.

Figure 11. Design Space Explorer II



2.5.1.2 Optimizing with Project Revisions

You can save multiple, named project revisions within your Quartus Prime project (**Project > Revisions**).

Each revision captures a unique set of project settings and constraints, but does not capture any logic design file changes. Use revisions to experiment with different settings while preserving the original. Optimize different revisions for various applications. Use revisions for the following:

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

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

2.5.1.3 Copying Your Project

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

The project copy includes all design files, `.qsf`(s), and project revisions. Use this technique to optimize project copies for different applications. For example, optimize one project to interface with a 32-bit data bus, and optimize a project copy to interface with a 64-bit data bus.

2.5.1.4 Copy (Back-Annotate) Compiler Assignments

The Compiler maps the elements of your design to specific device and resource during fitting. Following compilation processing, you can copy the Compiler's device and resource assignments to the `.qsf` to preserve that same implementation in subsequent compilations.

Click **Assignments ➤ Back-Annoate Assignments**to apply the device resource assignments to the `.qsf`. Select the back-annotation type in the **Back-annotation type** list.

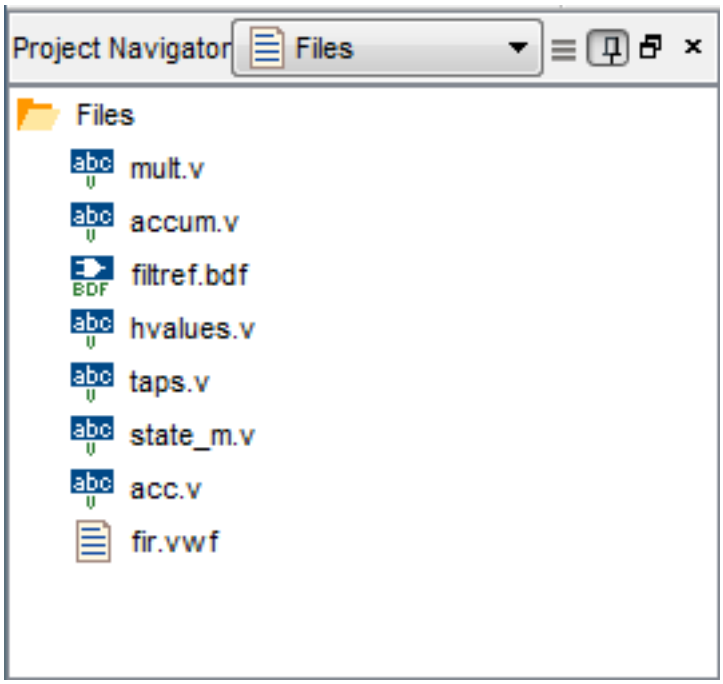
2.6 Managing Logic Design Files

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

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

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

Figure 12. Design and IP Files in Project Navigator



Right-click files in the Project Navigator to:

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

2.6.1 Including Design Libraries

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

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

Related Links

Design Library Migration Guidelines on page 82

2.6.1.1 Specifying Design Libraries

Follow these steps to specify project libraries from the GUI.

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

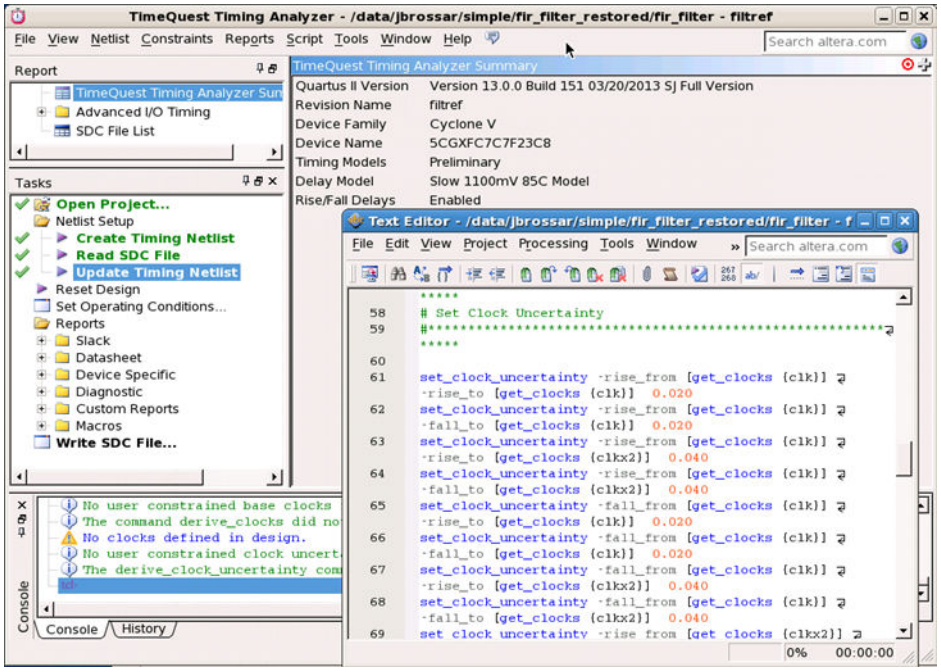
2.7 Managing Timing Constraints

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

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

Save the constraints you specify in the GUI in an industry-standard Synopsys Design Constraints File (`.sdc`). You can subsequently edit the text-based `.sdc` file directly. The order of the `.sdc` files in the `.sdc` is the order using the TimingQuest Timing Analyzer.

Figure 13. TimeQuest Timing Analyzer and SDC Syntax Example



2.8 Introduction to Intel® FPGA IP Cores

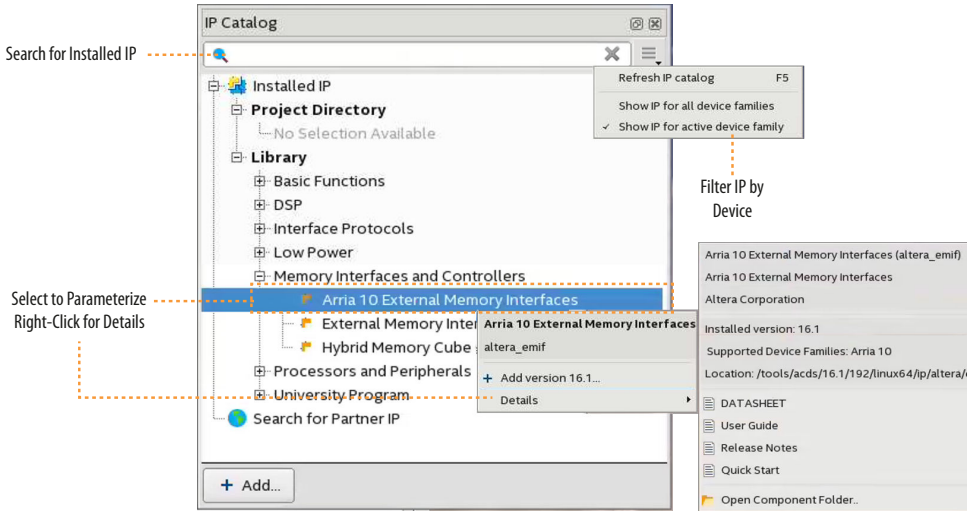
Intel and strategic IP partners offer a broad portfolio of configurable IP cores optimized for Intel FPGA devices.

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

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

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

Figure 14. IP Catalog



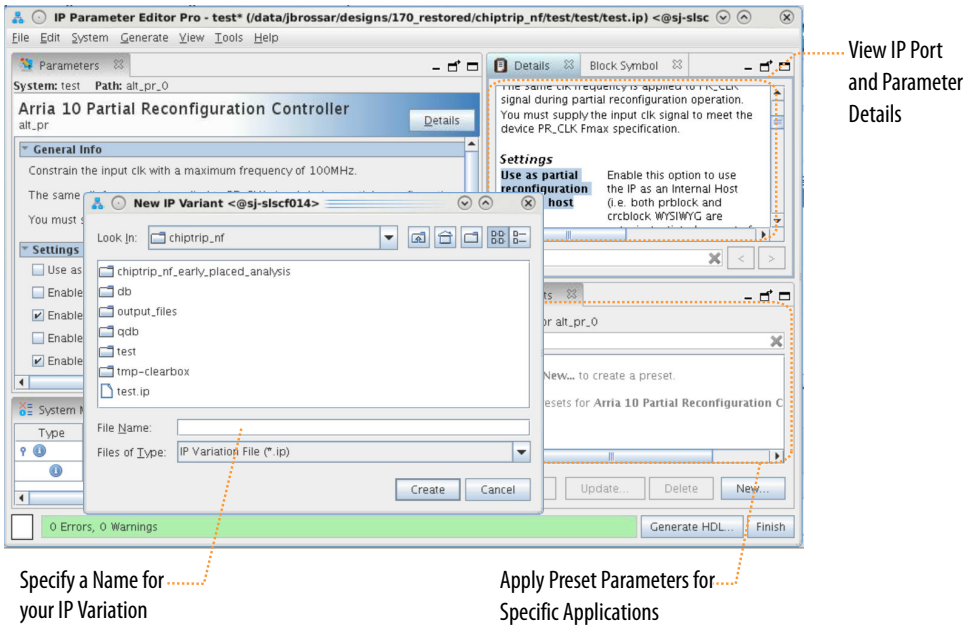
2.8.1 IP Catalog and Parameter Editor

The IP Catalog displays the IP cores available for your project. Use the following features of the IP Catalog to locate and customize an IP core:

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

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

Figure 15. IP Parameter Editor (Quartus Prime Pro Edition)



Related Links

Creating a System With Qsys Pro on page 299
Qsys Pro is a system integration tool included as part of the Quartus Prime software.

2.8.1.1 The Parameter Editor

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

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

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

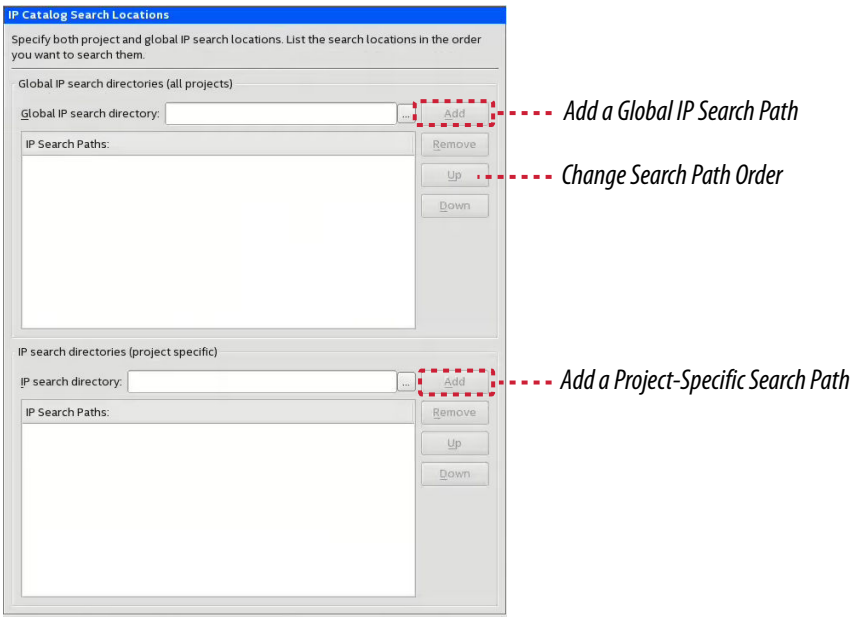
Related Links

- Creating a System with Qsys Pro
- Creating a System with Qsys

2.8.1.2 Adding IP Cores to IP Catalog

The IP Catalog automatically displays IP cores located in the project directory, in the default Quartus Prime installation directory, and in the IP search path.

Figure 16. Specifying IP Search Locations



The IP Catalog displays Quartus Prime IP components and Qsys Pro systems, third-party IP components, and any custom IP components that you include in the path. Use the **IP Search Path** option (**Tools ➤ Options**) to include custom and third-party IP components in the IP Catalog.

The Quartus Prime software searches the directories listed in the IP search path for the following IP core files:

- Component Description File (`_hw.tcl`)—defines a single IP core.
- IP Index File (`.ipx`)—each `.ipx` file indexes a collection of available IP cores. This file specifies the relative path of directories to search for IP cores. In general, `.ipx` files facilitate faster searches.

The Quartus Prime software searches some directories recursively and other directories only to a specific depth. When the search is recursive, the search stops at any directory that contains a `_hw.tcl` or `.ipx` file.

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

Table 14. IP Search Locations

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

If the Quartus Prime software recognizes two IP cores with the same name, the following search path precedence rules determine the resolution of files:

1. Project directory.
2. Project database directory.
3. Project IP search path specified in **IP Search Locations**, or with the SEARCH_PATH assignment for the current project revision.
4. Global IP search path specified in **IP Search Locations**, or with the SEARCH_PATH assignment in the `quartus2.ini` file.
5. Quartus software libraries directory, such as <*`Quartus Installation`*>`\libraries`.

Note: If you add an IP component to the search path, update the IP Catalog by clicking **Refresh IP Catalog** in the drop-down list. In Qsys and Qsys Pro, click **File ➤ Refresh System** to update the IP Catalog.

2.8.1.3 General Settings for IP

Use the following settings to control how the Quartus Prime software manages IP cores in your project.

Table 15. IP Core General Setting Locations

Setting Location	Description
Tools ➤ Options ➤ IP Settings Or **Tasks pane ➤ Settings ➤ IP Settings** (Pro Edition Only)	• Specify the **IP generation HDL preference**. The parameter editor generates the HDL you specify for IP variations. • Increase the **Maximum Qsys memory usage size** if you experience slow processing for large systems, or for out of memory errors. • Specify whether to **Automatically add Quartus Prime IP files** to all projects. Disable this option to manually add the IP files. • Use the **IP Regeneration Policy** setting to control when synthesis files regenerate for each IP variation. Typically, you **Always regenerate synthesis files for IP cores** after making changes to an IP variation.
Tools ➤ Options ➤ IP Catalog Search Locations Or **Tasks pane ➤ Settings ➤ IP Catalog Search Locations** (Pro Edition Only)	• Specify additional project and global IP search locations. The Quartus Prime software searches for IP cores in the project directory, in the Quartus Prime installation directory, and in the IP search path.

2.8.1.4 Installing and Licensing IP Cores

The Intel® Quartus Prime software installation includes the Intel FPGA IP library. This library provides useful IP core functions for your production use without the need for an additional license. Some IP cores in the library require that you purchase a separate license for production use. The OpenCore® feature allows evaluation of any Intel FPGA IP core in simulation and compilation in the Quartus Prime software. Upon satisfaction with functionality and performance, visit the Self Service Licensing Center to obtain a license number for any Intel FPGA product.

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

Figure 17. IP Core Installation Path

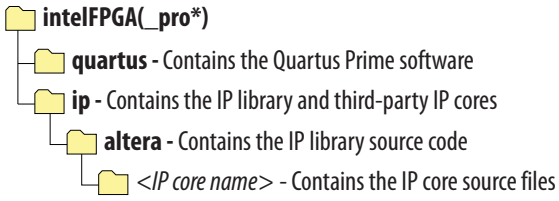


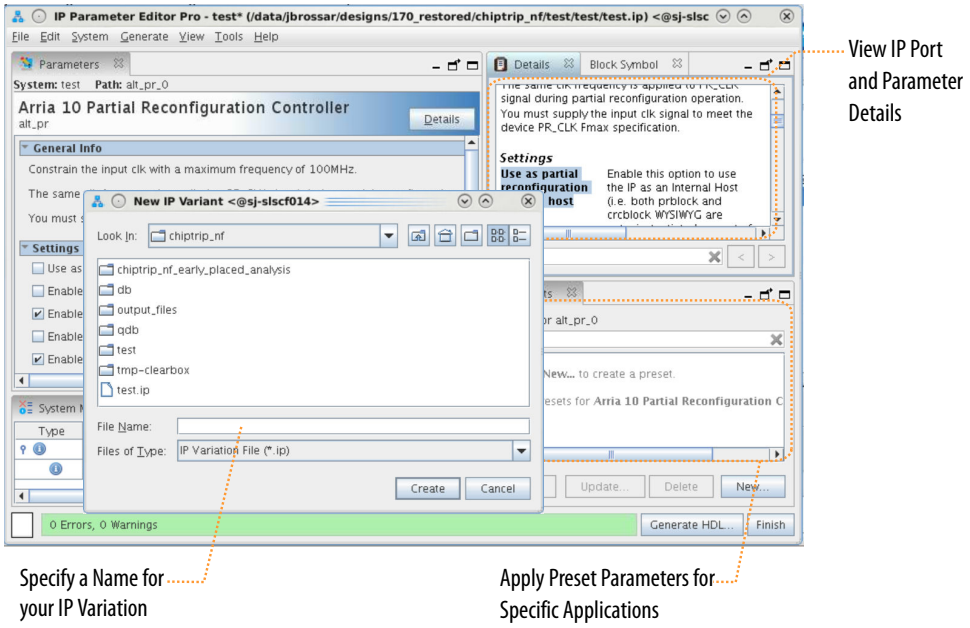
Table 16. IP Core Installation Locations

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

2.8.2 Generating IP Cores (Quartus Prime Pro Edition)

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

Figure 18. IP Parameter Editor (Quartus Prime Pro Edition)



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

1. Create or open a Quartus Prime project (`.qpf`) to contain the instantiated IP variation.
2. In the IP Catalog (**Tools ➤ IP Catalog**), locate and double-click the name of the IP core to customize. To locate a specific component, type some or all of the component's name in the IP Catalog search box. The New IP Variation window appears.
3. Specify a top-level name for your custom IP variation. Do not include spaces in IP variation names or paths. The parameter editor saves the IP variation settings in a file named <*your_ip*>`.ip`. Click **OK**. The parameter editor appears.
4. Set the parameter values in the parameter editor and view the block diagram for the component. The **Parameterization Messages** tab at the bottom displays any errors in IP parameters:
   - Optionally, select preset parameter values if provided for your IP core. Presets specify initial parameter values for specific applications.
   - Specify parameters defining the IP core functionality, port configurations, and device-specific features.
   - Specify options for processing the IP core files in other EDA tools.

   Note: Refer to your IP core user guide for information about specific IP core parameters.
5. Click **Generate HDL**. The **Generation** dialog box appears.
6. Specify output file generation options, and then click **Generate**. The synthesis and/or simulation files generate according to your specifications.
7. To generate a simulation testbench, click **Generate ➤ Generate Testbench System**. Specify testbench generation options, and then click **Generate**.

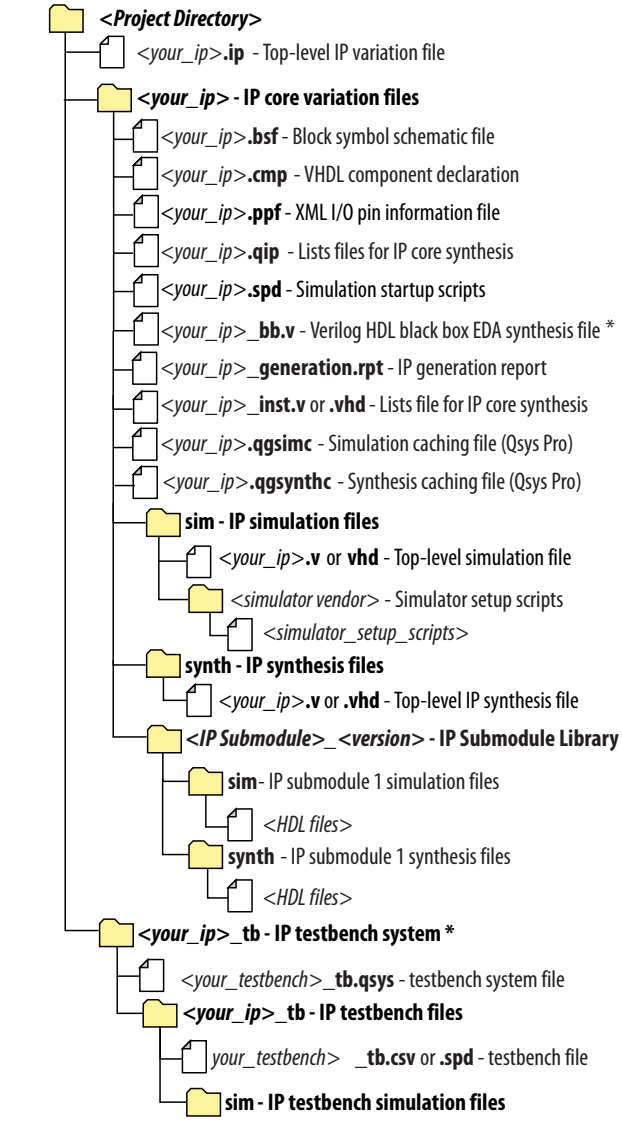
8. To generate an HDL instantiation template that you can copy and paste into your text editor, click **Generate ➤ Show Instantiation Template**.
9. Click **Finish**. Click **Yes** if prompted to add files representing the IP variation to your project.
10. After generating and instantiating your IP variation, make appropriate pin assignments to connect ports.

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

2.8.2.1 IP Core Generation Output (Quartus Prime Pro Edition)

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

Figure 19. Individual IP Core Generation Output (Quartus Prime Pro Edition)



* If supported and enabled for your IP core variation.

Table 17. Files Generated for IP Cores

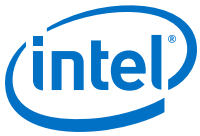
File Name	Description
<your_ip>.ip	Top-level IP variation file that contains the parameterization of an IP core in your project. If the IP variation is part of a Qsys Pro system, the parameter editor also generates a .qsys file.
<your_ip>.cmp	The VHDL Component Declaration (.cmp) file is a text file that contains local generic and port definitions that you use in VHDL design files.
<your_ip>_generation.rpt	IP or Qsys Pro generation log file. Displays a summary of the messages during IP generation.

continued...

File Name	Description
<*your_ip*>.qgsimc (Qsys Pro systems only)	Simulation caching file that compares the .qsys and .ip files with the current parameterization of the Qsys Pro system and IP core. This comparison determines if Qsys Pro can skip regeneration of the HDL.
<*your_ip*>.qgsynth (Qsys Pro systems only)	Synthesis caching file that compares the .qsys and .ip files with the current parameterization of the Qsys Pro system and IP core. This comparison determines if Qsys Pro can skip regeneration of the HDL.
<*your_ip*>.qip	Contains all information to integrate and compile the IP component.
<*your_ip*>.csv	Contains information about the upgrade status of the IP component.
<*your_ip*>.bsf	A symbol representation of the IP variation for use in Block Diagram Files (.bdf).
<*your_ip*>.spd	Required input file for ip-make-simscript to generate simulation scripts for supported simulators. The .spd file contains a list of files you generate for simulation, along with information about memories that you initialize.
<*your_ip*>.ppf	The Pin Planner File (.ppf) stores the port and node assignments for IP components you create for use with the Pin Planner.
<*your_ip*>_bb.v	Use the Verilog blackbox (_bb.v) file as an empty module declaration for use as a blackbox.
<*your_ip*>_inst.v or _inst.vhd	HDL example instantiation template. Copy and paste the contents of this file into your HDL file to instantiate the IP variation.
<*your_ip*>.regmap	If the IP contains register information, the Quartus Prime software generates the .regmap file. The .regmap file describes the register map information of master and slave interfaces. This file complements the .sopcinfo file by providing more detailed register information about the system. This file enables register display views and user customizable statistics in System Console.
<*your_ip*>.svd	Allows HPS System Debug tools to view the register maps of peripherals that connect to HPS within a Qsys Pro system. During synthesis, the Quartus Prime software stores the .svd files for slave interface visible to the System Console masters in the .sof file in the debug session. System Console reads this section, which Qsys Pro queries for register map information. For system slaves, Qsys Pro accesses the registers by name.
<*your_ip*>.v <*your_ip*>.vhd	HDL files that instantiate each submodule or child IP core for synthesis or simulation.
mentor/	Contains a ModelSim™ LNL script msim_setup.tcl to set up and run a simulation.
aldec/	Contains a Riviera*-PRO script rivierapro_setup.tcl to setup and run a simulation.
/synopsys/vcs /synopsys/vcsmx	Contains a shell script vcs_setup.sh to set up and run a VCS* simulation. Contains a shell script vcsmx_setup.sh and synopsys_sim.setup file to set up and run a VCS MX* simulation.
/cadence	Contains a shell script ncsim_setup.sh and other setup files to set up and run an NCSIM simulation.
/submodules	Contains HDL files for the IP core submodule.
<*IP submodule*>/	For each generated IP submodule directory, Qsys Pro generates /synth and /sim sub-directories.

2.8.2.2 Scripting IP Core Generation

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



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

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

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

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

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

Table 18. qsys-generate Command-Line Options

Option	Usage	Description
<1st arg file>	Required	Specifies the name of the `.qsys` system file to generate.
`--synthesis=<`*VERILOG*`|`*VHDL*`>`	Optional	Creates synthesis HDL files that Qsys Pro uses to compile the system in a Quartus Prime project. Specify the preferred generation language for the top-level RTL file for the generated Qsys Pro system. The default value is *VERILOG*.
`--block-symbol-file`	Optional	Creates a Block Symbol File (**.bsf**) for the Qsys Pro system.
`--greybox`	Optional	If you are synthesizing your design with a third-party EDA synthesis tool, generate a netlist for the synthesis tool to estimate timing and resource usage for this design.
`--ipxact`	Optional	If you set this option to true, Qsys Pro renders the post-generation system as an IPXACT-compatible component description.
`--simulation=<`*VERILOG*`|`*VHDL*`>`	Optional	Creates a simulation model for the Qsys Pro system. The simulation model contains generated HDL files for the simulator, and may include simulation-only features. Specify the preferred simulation language. The default value is *VERILOG*.
`--testbench=<`*SIMPLE*`|`*STANDARD*`>`	Optional	Creates a testbench system that instantiates the original system, adding bus functional models (BFMs) to drive the top-level interfaces. When you generate the system, the BFMs interact with the system in the simulator. The default value is *STANDARD*.
`--testbench-simulation=<`*VERILOG*`|`*VHDL*`>`	Optional	After you create the testbench system, create a simulation model for the testbench system. The default value is *VERILOG*.
`--example-design=<`*value*`>`	Optional	Creates example design files. For example, `--example-design` or `--example-design=all`. The default is *All*, which generates example designs for all instances. Alternatively, choose specific filesets based on instance name and fileset name. For example `--example-design=instance0.example_design1,instance1.example_design 2`. Specify an output directory for the example design files creation.
`--search-path=<`*value*`>`	Optional	If you omit this command, Qsys Pro uses a standard default path. If you provide this command, Qsys Pro searches a comma-separated list of paths. To include the standard path in your replacement, use "$", for example, `"/extra/dir,$"`.
`--family=<`*value*`>`	Optional	Sets the device family name.

continued...

Option	Usage	Description
`--part=<value>`	Optional	Sets the device part number. If set, this option overrides the `--family` option.
`--upgrade-variation-file`	Optional	If you set this option to true, the file argument for this command accepts a `.v` file, which contains a IP variant. This file parameterizes a corresponding instance in a Qsys Pro system of the same name.
`--upgrade-ip-cores`	Optional	Enables upgrading all the IP cores that support upgrade in the Qsys Pro system.
`--clear-output-directory`	Optional	Clears the output directory corresponding to the selected target, that is, simulation or synthesis.
`--jvm-max-heap-size=<value>`	Optional	The maximum memory size that Qsys Pro uses when running `qsys-generate`. You specify the value as `<size><unit>`, where `unit` is m (or M) for multiples of megabytes or g (or G) for multiples of gigabytes. The default value is 512m.
`--help`	Optional	Displays help for `--qsys-generate`.

2.8.3 Modifying an IP Variation

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

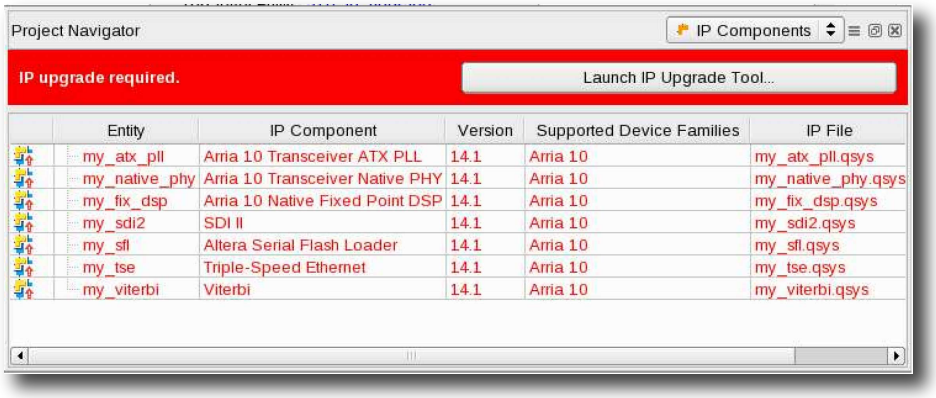
Table 19. Modifying an IP Variation

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

2.8.4 Upgrading IP Cores

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

Figure 20. IP Upgrade Alert in Project Navigator



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

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

Table 20. IP Core Upgrade Status

IP Core Status	Description
IP Upgraded 	Indicates that your IP variation uses the latest version of the IP core.
IP Upgrade Optional 	Indicates that upgrade is optional for this IP variation in the current version of the Quartus Prime software. Optionally, upgrade this IP variation to take advantage of the latest development of this IP core. Retain previous IP core characteristics by declining to upgrade. Refer to the Description for details about IP core version differences. If you do not upgrade the IP, the IP variation synthesis and simulation files remain unchanged, and you cannot modify parameters until upgrading.
IP Upgrade Required 	Indicates that you must upgrade the IP variation before compiling in the current version of the Quartus Prime software. Refer to the Description for details about IP core version differences.
IP Upgrade Unsupported 	Indicates that Quartus Prime software does not support upgrade of the IP variation due to incompatibility in the current software version. The Quartus Prime software prompts you to replace the unsupported IP core with equivalent IP core from the IP Catalog. Refer to the Description for details about IP core version differences and links to Release Notes.

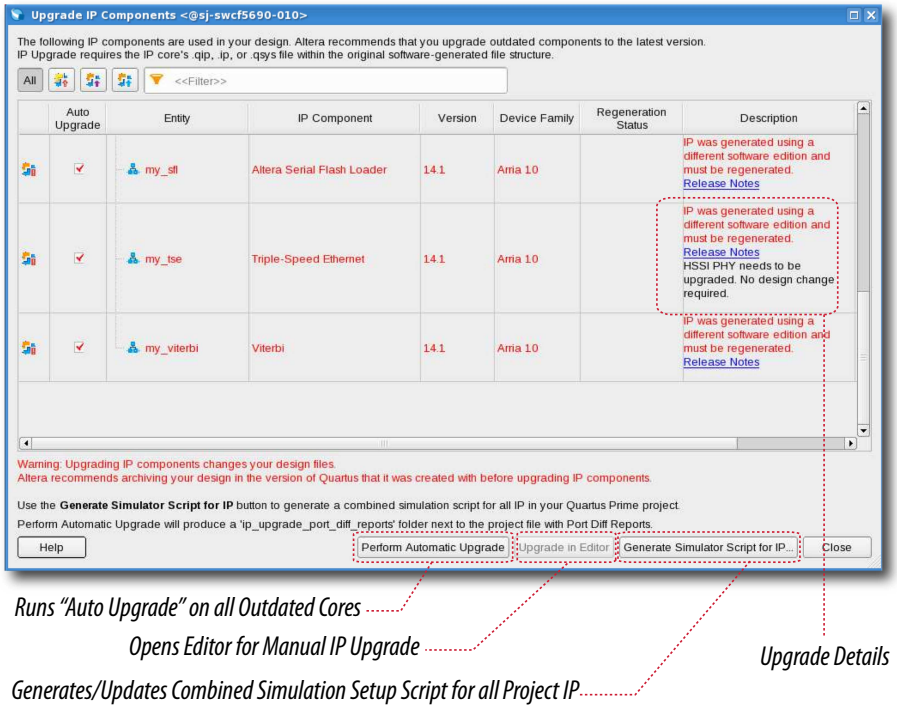
continued...

IP Core Status	Description
IP End of Life 	Indicates that Intel designates the IP core as end-of-life status. You may or may not be able to edit the IP core in the parameter editor. Support for this IP core discontinues in future releases of the Quartus Prime software.
IP Upgrade Mismatch Warning 	Provides warning of non-critical IP core differences in migrating IP to another device family.

Follow these steps to upgrade IP cores:

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

Figure 21. Upgrading IP Cores



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

Related Links

Intel FPGA IP Core Release Notes

2.8.4.1 Upgrading IP Cores at Command-Line

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

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

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

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

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

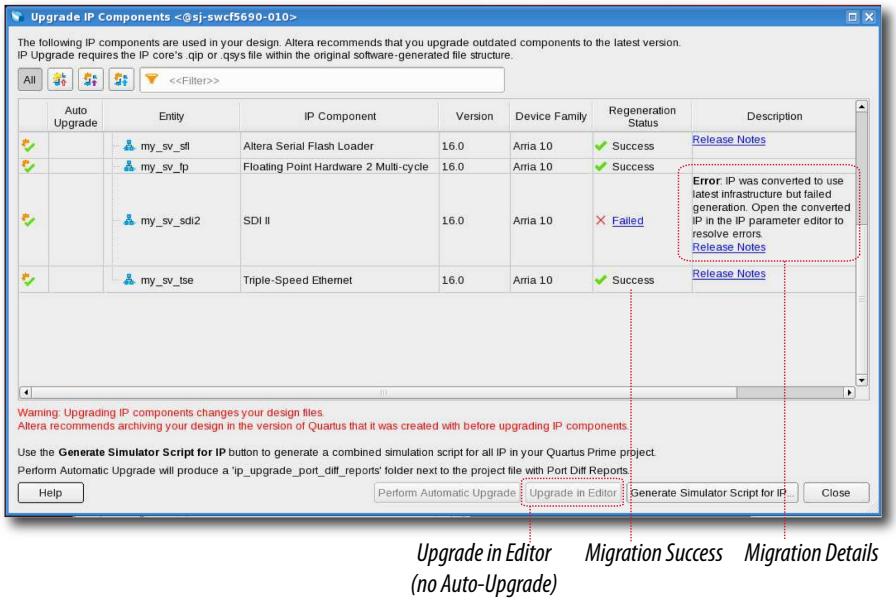
```
quartus_sh –ip_upgrade –variation_files “<my_ip1>.<qsys,.v, .vhd>>;
<my_ip_filepath/my_ip2>.<hdl>”  <quartus_project>

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

2.8.4.2 Migrating IP Cores to a Different Device

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

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

Figure 22. IP Core Device Migration



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

Related Links

Intel FPGA IP Release Notes

2.8.4.3 Troubleshooting IP or Qsys Pro System Upgrade

The **Upgrade IP Components** dialog box reports the version and status of each IP core and Qsys Pro system following upgrade or migration.

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

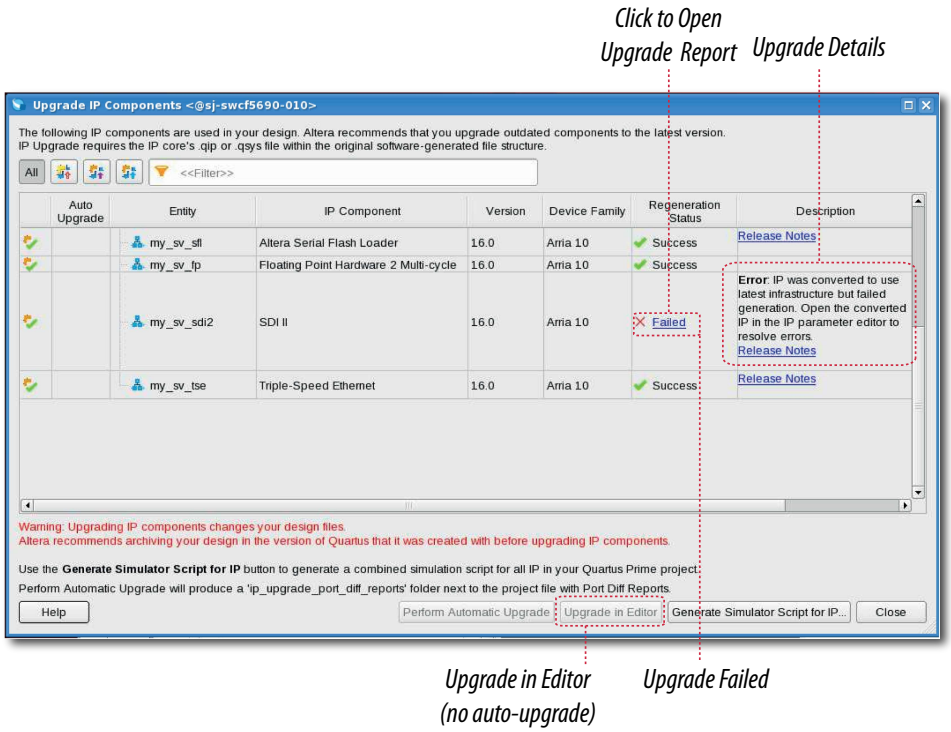
Note: Do not use spaces in IP variation names or paths.

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

Table 21. IP Upgrade Error Information

Upgrade IP Components Field	Description
Regeneration Status	Displays the "Success" or "Failed" status of each upgrade or migration. Click the status of any upgrade that fails to open the **IP Upgrade Report**.
Version	Dynamically updates the version number when upgrade is successful. The text is red when the IP requires upgrade.
Device Family	Dynamically updates to the new device family when migration is successful. The text is red when the IP core requires upgrade.
Description	Summarizes IP release information and displays corrective action for resolving upgrade or migration failures. Click the **Release Notes** link for the latest known issues about the IP core.
Perform Automatic Upgrade	Runs automatic upgrade on all IP cores that support auto upgrade. Also, automatically generates a *<Project Directory>*/ip_upgrade_port_diff_report report for IP cores or Qsys Pro systems that fail upgrade. Review these reports to determine any port differences between the current and previous IP core version.

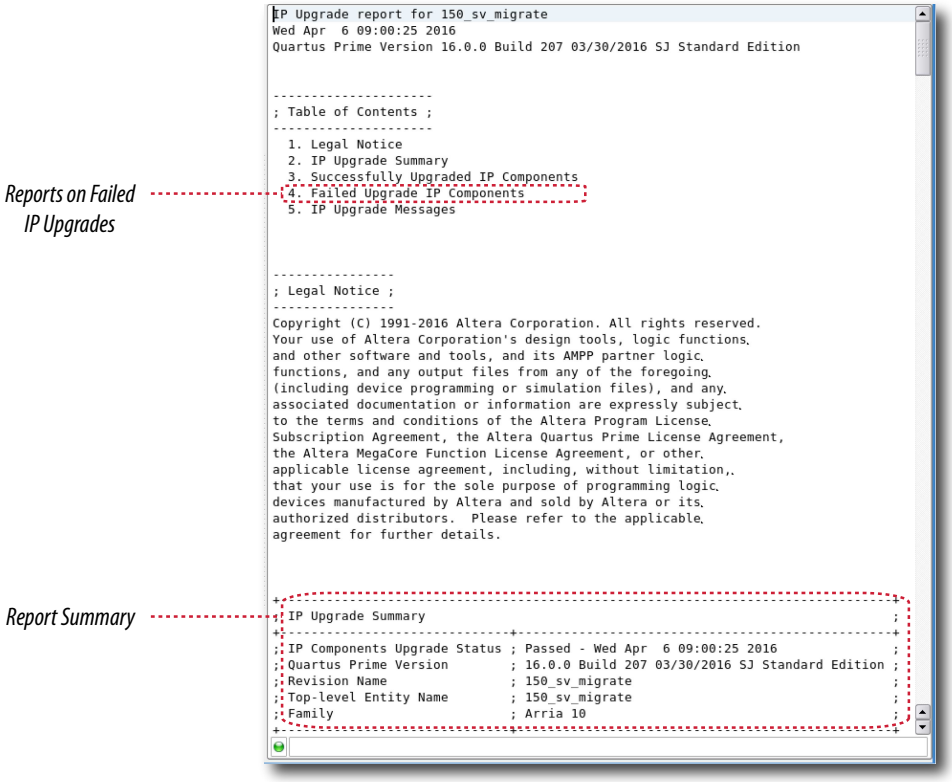
Figure 23. Resolving Upgrade Errors



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

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

Figure 24. IP Upgrade Report



2.8.5 Simulating Intel FPGA IP Cores

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

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

2.8.5.1 Generating IP Simulation Files

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

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

Table 22. Intel FPGA IP Simulation Files

File Type	Description	File Name

Simulator setup scripts	Vendor-specific scripts to compile, elaborate, and simulate Intel FPGA IP models and simulation model library files. Optionally, generate a simulator setup script for each vendor that combines the individual IP core scripts into one file. Source the combined script from your top-level simulation script to eliminate script maintenance.	`<my_dir>/aldec/rivierapro_setup.tcl`
		`<my_dir>/cadence/ncsim_setup.sh`
		`<my_dir>/mentor/msim_setup.tcl`
		`<my_dir>/synopsys/vcs/vcs_setup.sh`

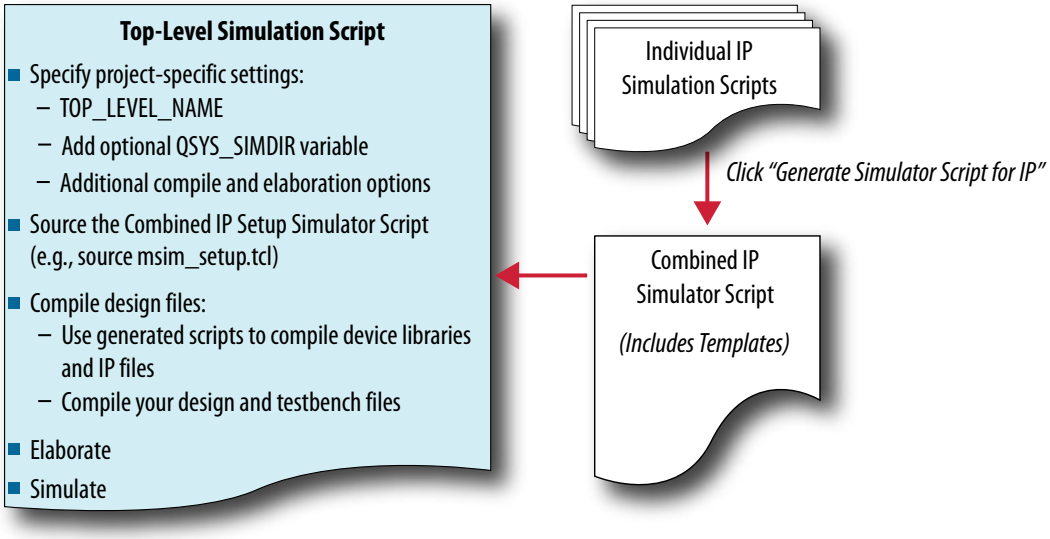
Note: Intel FPGA IP cores support a variety of cycle-accurate simulation models, including simulation-specific IP functional simulation models and encrypted RTL models, and plain text RTL models. The models support fast functional simulation of your IP core instance using industry-standard VHDL or Verilog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

2.8.5.2 Scripting IP Simulation

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

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

Figure 25. Incorporating Generated Simulator Setup Scripts into a Top-Level Simulation Script



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

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

2.8.5.2.1 Generating a Combined Simulator Setup Script

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

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

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

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

1. Generate, regenerate, or upgrade one or more IP core. Refer to *Generating IP Cores* or *Upgrading IP Cores*.
2. Click **Tools ➤ Generate Simulator Setup Script for IP** (or run the `ip-setup-simulation` utility). Specify the **Output Directory** and library compilation options. Click **OK** to generate the file. By default, the files generate into the */<project directory>/<simulator>/* directory using relative paths.
3. To incorporate the generated simulator setup script into your top-level simulation script, refer to the template section in the generated simulator setup script as a guide to creating a top-level script:
   a. Copy the specified template sections from the simulator-specific generated scripts and paste them into a new top-level file.
   b. Remove the comments at the beginning of each line from the copied template sections.
   c. Specify the customizations you require to match your design simulation requirements, for example:
      - Specify the TOP_LEVEL_NAME variable to the design's simulation top-level file. The top-level entity of your simulation is often a testbench that instantiates your design. Then, your design instantiates IP cores and/or Qsys or Qsys Pro systems. Set the value of TOP_LEVEL_NAME to the top-level entity.
      - If necessary, set the QSYS_SIMDIR variable to point to the location of the generated IP simulation files.
      - Compile the top-level HDL file (e.g. a test program) and all other files in the design.
      - Specify any other changes, such as using the `grep` command-line utility to search a transcript file for error signatures, or e-mail a report.
4. Re-run **Tools ➤ Generate Simulator Setup Script for IP** (or `ip-setup-simulation`) after regeneration of an IP variation.

Table 23. Simulation Script Utilities

Utility	Syntax

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

`ip-make-simscript` generates a combined simulation script for all IP cores that you specify on the command line. Specify one or more `.spd` files and an output directory in the command. Running the script compiles IP simulation models into various simulation libraries.	`ip-make-simscript`
	` --spd=<ipA.spd,ipB.spd>`
	` --output-directory=<directory>`

The following sections provide step-by-step instructions for sourcing each simulator setup script in your top-level simulation script.

Sourcing Aldec* Simulator Setup Scripts

Follow these steps to incorporate the generated Aldec simulation scripts into a top-level project simulation script.

1. The generated simulation script contains the following template lines. Cut and paste these lines into a new file. For example, `sim_top.tcl`.

```
# # Start of template
# # If the copied and modified template file is "aldec.do", run it as:
# # vsim -c -do aldec.do
# #
# # Source the generated sim script
# source rivierapro_setup.tcl
# # Compile eda/sim_lib contents first
# dev_com
# # Override the top-level name (so that elab is useful)
# set TOP_LEVEL_NAME top
# # Compile the standalone IP.
# com
# # Compile the user top-level
# vlog -sv2k5 ../../top.sv
# # Elaborate the design.
# elab
# # Run the simulation
# run
# # Report success to the shell
# exit -code 0
# # End of template
```

2. Delete the first two characters of each line (comment and space):

```
# Start of template
# If the copied and modified template file is "aldec.do", run it as:
# vsim -c -do aldec.do
#
# Source the generated sim script source rivierapro_setup.tcl
# Compile eda/sim_lib contents first dev_com
# Override the top-level name (so that elab is useful)
set TOP_LEVEL_NAME top
# Compile the standalone IP.
com
```

```
# Compile the user top-level vlog -sv2k5 ../../top.sv
# Elaborate the design.
elab
# Run the simulation
run
# Report success to the shell
exit -code 0
# End of template
```

3. Modify the TOP_LEVEL_NAME and compilation step appropriately, depending on the simulation's top-level file. For example:

```
set TOP_LEVEL_NAME sim_top
 vlog –sv2k5 ../../sim_top.sv
```

4. If necessary, add the QSYS_SIMDIR variable to point to the location of the generated IP simulation files. Specify any other changes that you require to match your design simulation requirements. The scripts offer variables to set compilation or simulation options. Refer to the generated script for details.
5. Run the new top-level script from the generated simulation directory:

```
vsim –c –do <path to sim_top>.tcl
```

Sourcing Cadence* Simulator Setup Scripts

Follow these steps to incorporate the generated Cadence IP simulation scripts into a top-level project simulation script.

1. The generated simulation script contains the following template lines. Cut and paste these lines into a new file. For example, ncsim.sh.

```
# # Start of template
# # If the copied and modified template file is "ncsim.sh", run it as:
# # ./ncsim.sh
# #
# # Do the file copy, dev_com and com steps
# source ncsim_setup.sh \
# SKIP_ELAB=1 \
# SKIP_SIM=1
#
# # Compile the top level module
# ncvlog -sv "$QSYS_SIMDIR/../top.sv"
#
# # Do the elaboration and sim steps
# # Override the top-level name
# # Override the user-defined sim options, so the simulation
# # runs forever (until $finish()).
# source ncsim_setup.sh \
# SKIP_FILE_COPY=1 \
# SKIP_DEV_COM=1 \
# SKIP_COM=1 \
# TOP_LEVEL_NAME=top \
# USER_DEFINED_SIM_OPTIONS=""
# # End of template
```

2. Delete the first two characters of each line (comment and space):

```
# Start of template
# If the copied and modified template file is "ncsim.sh", run it as:
# ./ncsim.sh
#
# Do the file copy, dev_com and com steps
```

```
source ncsim_setup.sh \
SKIP_ELAB=1 \
SKIP_SIM=1
# Compile the top level module
ncvlog -sv "$QSYS_SIMDIR/../top.sv"
# Do the elaboration and sim steps
# Override the top-level name
# Override the user-defined sim options, so the simulation
# runs forever (until $finish()).
source ncsim_setup.sh \
SKIP_FILE_COPY=1 \
SKIP_DEV_COM=1 \
SKIP_COM=1 \
TOP_LEVEL_NAME=top \
USER_DEFINED_SIM_OPTIONS=""
# End of template
```

3. Modify the TOP_LEVEL_NAME and compilation step appropriately, depending on the simulation's top-level file. For example:

```
TOP_LEVEL_NAME=sim_top \
 ncvlog -sv "$QSYS_SIMDIR/../top.sv"
```

4. If necessary, add the QSYS_SIMDIR variable to point to the location of the generated IP simulation files. Specify any other changes that you require to match your design simulation requirements. The scripts offer variables to set compilation or simulation options. Refer to the generated script for details.
5. Run the resulting top-level script from the generated simulation directory by specifying the path to `ncsim.sh`.

Sourcing ModelSim* Simulator Setup Scripts

Follow these steps to incorporate the generated ModelSim IP simulation scripts into a top-level project simulation script.

1. The generated simulation script contains the following template lines. Cut and paste these lines into a new file. For example, `sim_top.tcl`.

```
# # Start of template
# # If the copied and modified template file is "mentor.do", run it
# # as: vsim -c -do mentor.do
# #
# # Source the generated sim script
# source msim_setup.tcl
# # Compile eda/sim_lib contents first
# dev_com
# # Override the top-level name (so that elab is useful)
# set TOP_LEVEL_NAME top
# # Compile the standalone IP.
# com
# # Compile the user top-level
# vlog -sv ../../top.sv
# # Elaborate the design.
# elab
# # Run the simulation
# run -a
```

```
# # Report success to the shell
# exit -code 0
# # End of template
```

2. Delete the first two characters of each line (comment and space):

```
# Start of template
# If the copied and modified template file is "mentor.do", run it
# as: vsim -c -do mentor.do
#
# Source the generated sim script source msim_setup.tcl
# Compile eda/sim_lib contents first
dev_com
# Override the top-level name (so that elab is useful)
set TOP_LEVEL_NAME top
# Compile the standalone IP.
com
# Compile the user top-level vlog -sv ../../top.sv
# Elaborate the design.
elab
# Run the simulation
run -a
# Report success to the shell
exit -code 0
# End of template
```

3. Modify the TOP_LEVEL_NAME and compilation step appropriately, depending on the simulation's top-level file. For example:

```
set TOP_LEVEL_NAME sim_top vlog -sv ../../sim_top.sv
```

4. If necessary, add the QSYS_SIMDIR variable to point to the location of the generated IP simulation files. Specify any other changes required to match your design simulation requirements. The scripts offer variables to set compilation or simulation options. Refer to the generated script for details.
5. Run the resulting top-level script from the generated simulation directory:

```
vsim –c –do <path to sim_top>.tcl
```

Sourcing VCS* Simulator Setup Scripts

Follow these steps to incorporate the generated Synopsys VCS simulation scripts into a top-level project simulation script.

1. The generated simulation script contains these template lines. Cut and paste the lines preceding the "helper file" into a new executable file. For example, synopsys_vcs.f.

```
# # Start of template
# # If the copied and modified template file is "vcs_sim.sh", run it
# # as: ./vcs_sim.sh
# #
# # Override the top-level name
# # specify a command file containing elaboration options
# # (system verilog extension, and compile the top-level).
# # Override the user-defined sim options, so the simulation
# # runs forever (until $finish()).
# source vcs_setup.sh \
# TOP_LEVEL_NAME=top \
# USER_DEFINED_ELAB_OPTIONS="'-f ../../../synopsys_vcs.f'" \
# USER_DEFINED_SIM_OPTIONS=""
#
```

```
# # helper file: synopsys_vcs.f
# +systemverilogext+.sv
# ../../../top.sv
# # End of template
```

2. Delete the first two characters of each line (comment and space) for the `vcs.sh` file, as shown below:

```
# Start of template
# If the copied and modified template file is "vcs_sim.sh", run it
# as: ./vcs_sim.sh
#
# Override the top-level name
# specify a command file containing elaboration options
# (system verilog extension, and compile the top-level).
# Override the user-defined sim options, so the simulation
# runs forever (until $finish()).
source vcs_setup.sh \
TOP_LEVEL_NAME=top \
USER_DEFINED_ELAB_OPTIONS="'-f ../../../synopsys_vcs.f'" \
USER_DEFINED_SIM_OPTIONS=""
```

3. Delete the first two characters of each line (comment and space) for the `synopsys_vcs.f` file, as shown below:

```
# helper file: synopsys_vcs.f
 +systemverilogext+.sv
 ../../../top.sv
# End of template
```

4. Modify the TOP_LEVEL_NAME and compilation step appropriately, depending on the simulation's top-level file. For example:

```
TOP_LEVEL_NAME=sim_top \
```

5. If necessary, add the QSYS_SIMDIR variable to point to the location of the generated IP simulation files. Specify any other changes required to match your design simulation requirements. The scripts offer variables to set compilation or simulation options. Refer to the generated script for details.
6. Run the resulting top-level script from the generated simulation directory by specifying the path to `vcs_sim.sh`.

Sourcing VCS* MX Simulator Setup Scripts

Follow these steps to incorporate the generated Synopsys VCS MX simulation scripts for use in top-level project simulation scripts.

1. The generated simulation script contains these template lines. Cut and paste the lines preceding the "helper file" into a new executable file. For example, `vcsmx.sh`.

```
# # Start of template
# # If the copied and modified template file is "vcsmx_sim.sh", run
# # it as: ./vcsmx_sim.sh
# #
# # Do the file copy, dev_com and com steps
# source vcsmx_setup.sh \
# SKIP_ELAB=1 \

# SKIP_SIM=1
#
```

```
# # Compile the top level module vlogan +v2k
     +systemverilogext+.sv "$QSYS_SIMDIR/../top.sv"

# # Do the elaboration and sim steps
# # Override the top-level name
# # Override the user-defined sim options, so the simulation runs
# # forever (until $finish()).
# source vcsmx_setup.sh \
# SKIP_FILE_COPY=1 \
# SKIP_DEV_COM=1 \
# SKIP_COM=1 \
# TOP_LEVEL_NAME="'-top top'" \
# USER_DEFINED_SIM_OPTIONS=""
# # End of template
```

2. Delete the first two characters of each line (comment and space), as shown below:

```
# Start of template
# If the copied and modified template file is "vcsmx_sim.sh", run
# it as: ./vcsmx_sim.sh
#
# Do the file copy, dev_com and com steps
source vcsmx_setup.sh \
SKIP_ELAB=1 \
SKIP_SIM=1

# Compile the top level module
vlogan +v2k +systemverilogext+.sv "$QSYS_SIMDIR/../top.sv"

# Do the elaboration and sim steps
# Override the top-level name
# Override the user-defined sim options, so the simulation runs
# forever (until $finish()).
source vcsmx_setup.sh \
SKIP_FILE_COPY=1 \
SKIP_DEV_COM=1 \
SKIP_COM=1 \
TOP_LEVEL_NAME="'-top top'" \
USER_DEFINED_SIM_OPTIONS=""
# End of template
```

3. Modify the TOP_LEVEL_NAME and compilation step appropriately, depending on the simulation's top-level file. For example:

```
TOP_LEVEL_NAME=”-top sim_top’” \
```

4. Make the appropriate changes to the compilation of the your top-level file, for example:

```
vlogan +v2k +systemverilogext+.sv "$QSYS_SIMDIR/../sim_top.sv"
```

5. If necessary, add the QSYS_SIMDIR variable to point to the location of the generated IP simulation files. Specify any other changes required to match your design simulation requirements. The scripts offer variables to set compilation or simulation options. Refer to the generated script for details.
6. Run the resulting top-level script from the generated simulation directory by specifying the path to `vcsmx_sim.sh`.

2.8.6 Synthesizing IP Cores in Other EDA Tools

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

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

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

2.8.7 Instantiating IP Cores in HDL

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

2.8.7.1 Example Top-Level Verilog HDL Module

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

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

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

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

2.8.7.2 Example Top-Level VHDL Module

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

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

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

```
              result      : out std_logic_vector(31 downto 0));
end entity;

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

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

inst1 : altfp_mult
        generic map    (
                   pipeline => 11,
                   width_exp => 8,
                   width_man => 23,
                   exception_handling => "no")
        port map (
                   dataa => wire_dataa,
                   datab => datab,
                   clock => clock,
                   result => result);
end arch_MF_top;
```

2.8.8 Support for IP Core Encryption with the IEEE 1735 Standard

The Quartus Prime Pro Edition software supports the IEEE1735 v1 encryption standard for IP core decryption. The Quartus Prime Standard Edition software does not support this feature.

When you add the following Verilog or VHDL pragma to your RTL, along with the public key, the Quartus Prime software uses the key to decrypt the IP core. To use this feature, use a simulation or synthesis tool that supports the IEEE1735 standard.

Verilog/SystemVerilog Encryption Pragma:

```
`pragma protect key_keyowner = “Intel Corporation”
`pragma protect key_method = “rsa”
`pragma protect key_keyname = “Altera Key1”
`pragma protect key_block
<Encrypted session key>
```

VHDL Encryption Pragma:

```
`protect key_keyowner = “Intel Corporation”
`protect key_method = “rsa”
`protect key_keyname = “Altera Key1”
`protect key_block
<Encrypted session key>
```

For all languages, include the key value that is available from your sales representative or FAE.

Related Links

myAltera.com

2.9 Integrating Other EDA Tools

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

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

- Compile all RTL and gate-level simulation model libraries for your device, simulator, and design language automatically (**Tools > Launch Simulation Library Compiler**).
- Include files generated by other EDA design entry or synthesis tools in your project as synthesized design files (**Project > Add/Remove File from Project**) .
- Automatically generate optional files for board-level verification (**Assignments > Settings > EDA Tool Settings**).

2.10 Managing Team-based Projects

The Quartus Prime software supports multiple designers, design iterations, and platforms. Use the following techniques to preserve and track project changes in a team-based environment. These techniques may also be helpful for individual designers.

Related Links

- Using External Revision Control on page 80
- Migrating Projects Across Operating Systems on page 81

2.10.1 Preserving Compilation Results

The Quartus Prime software allows you to transfer your compiled databases from one version of the software to a newer version of the software.

You can export the results of compilation at various stages of the compilation flow, such as synthesis, planned, early place, place, route, and finalize snapshots. A *snapshot* is the compilation output of a compiler stage. Import allows you to restore the preserved compilation database and run subsequent stages in the compiler flow.

Export the compilation snapshot by clicking **Project ➤ Export Design**. The exported files are stored in a file with a `.qdb` extension. Import the snapshot with **Project ➤ Import Design**.

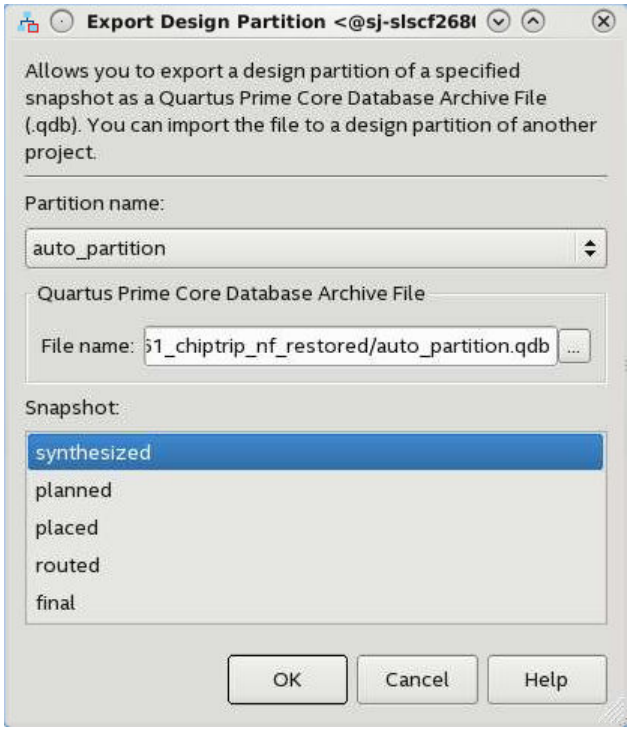
2.10.1.1 Exporting a Design Partition

A snapshot preserves the results of each compilation stage. To reuse the snapshot in another project, export the snapshot as a design partition.

Follow these steps to export a snapshot as a design partition:

1. Run Analysis & Synthesis or any stage of the Fitter on your design.
2. Click **Project ➤ Export Design Partition**.
3. Select the **Partition Name** of the entity for export.
4. Select the compilation **Snapshot** for export. The Compiler exports the snapshot as `<project>/<partition>.qdb`
5. To import the exported partition into another project, open the project and click **Project ➤ Import Design**. Specify the snapshot `.qdb` file.

Figure 26. Export Snapshot Partition



Related Links

Block-Level Design Flows

2.10.2 Factors Affecting Compilation Results

Various changes to project settings, hardware, or software can impact compilation results.

- Project Files—project settings (`.qsf`, `quartus2.ini`), design files, and timing constraints (`.sdc`). Any setting that changes the number of processors during compilation can impact compilation results.
- Hardware—CPU architecture, not including hard disk or memory size differences. Windows XP x32 results are not identical to Windows XP x64 results. Linux x86 results is not identical to Linux x86_64.
- Quartus Prime Software Version—including build number and installed patches. Click **Help > About** to obtain this information.
- Operating System—Windows or Linux operating system, excluding version updates. For example, Windows XP, Windows Vista, and Windows 7 results are identical. Similarly, Linux RHEL, CentOS 4, and CentOS 5 results are identical.

Related Links

- Design Planning for Partial Reconfiguration
- Power-Up Level

2.10.3 Migrating Compilation Results Across Quartus Prime Software Versions

View basic information about your project in the Project Navigator and Compilation Dashboard.

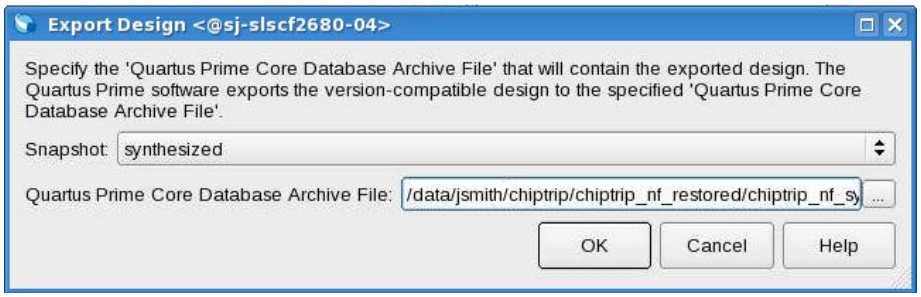
To preserve compilation results for migration to a newer version of the Quartus Prime software, export a version-compatible database file, and then import it into the later version of the Quartus Prime software.

2.10.3.1 Exporting the Results Database

Follow these steps to save the compilation results in a version-compatible format for import to a different version of the Quartus Prime software.

1. Open the project for exporting the compilation results in the Quartus Prime software.
2. Generate the project database and netlist with one of the following:
   - Click **Processing ➤ Start ➤ Start Analysis & Synthesis** to generate a post-synthesis netlist.
   - Click **Processing ➤ Start Compilation** to generate a post-fit netlist.
3. Click **Project ➤ Export Design**. Select the **Snapshot** for export. A **Quartus Prime Core Database Archive File** (`.qdb`) preserves the database. You can select one of the following **Snapshot**s:
   - **synthesized**—represents the output of analysis & synthesis.
   - **final**—represents the output of the Fitter.

Figure 27. Export Design Dialog Box



2.10.3.2 Importing the Results Database

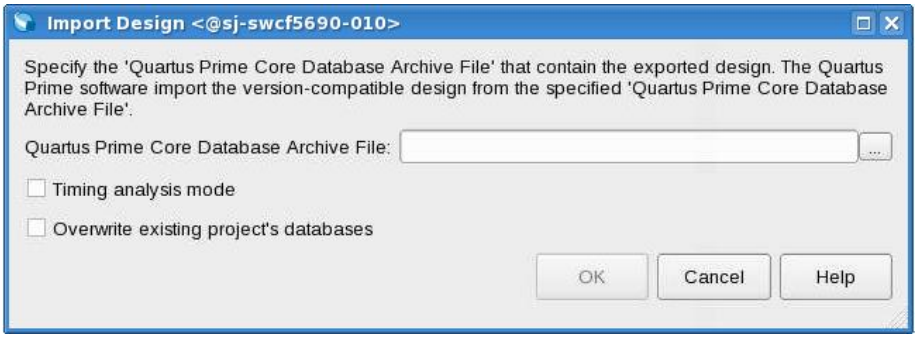
Follow these steps to import the compilation results from a previous version of the Quartus Prime software to another version of the software.

1. In a newer version of the Quartus Prime software, click **New Project Wizard** and create a new project with the same top-level design entity name as the database.
2. Click **Project ➤ Import Design** and specify the **Quartus Prime Core Database Archive File** that contains the exported results.

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

   The **Overwrite existing project's databases** option removes all prior compilation databases from the current project before importing the specified **Core Database Archive File**.

Figure 28. Import Design Dialog Box



2.10.4 Archiving Projects

Optionally save the elements of a project in a single, compressed Quartus Prime Archive File (`.qar`) by clicking **Project > Archive Project**.

The `.qar` preserves logic design, project, and settings files required to restore the project.

Use this technique to share projects between designers, or to transfer your project to a new version of the Quartus Prime software, or to Intel support. Optionally add compilation results, Qsys Pro system files, and third-party EDA tool files to the archive. If you restore the archive in a different version of the Quartus Prime software, you must include the original `.qdf` in the archive to preserve original compilation results.

2.10.4.1 Manually Adding Files To Archives

Follow these steps to add files to an archive manually.

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

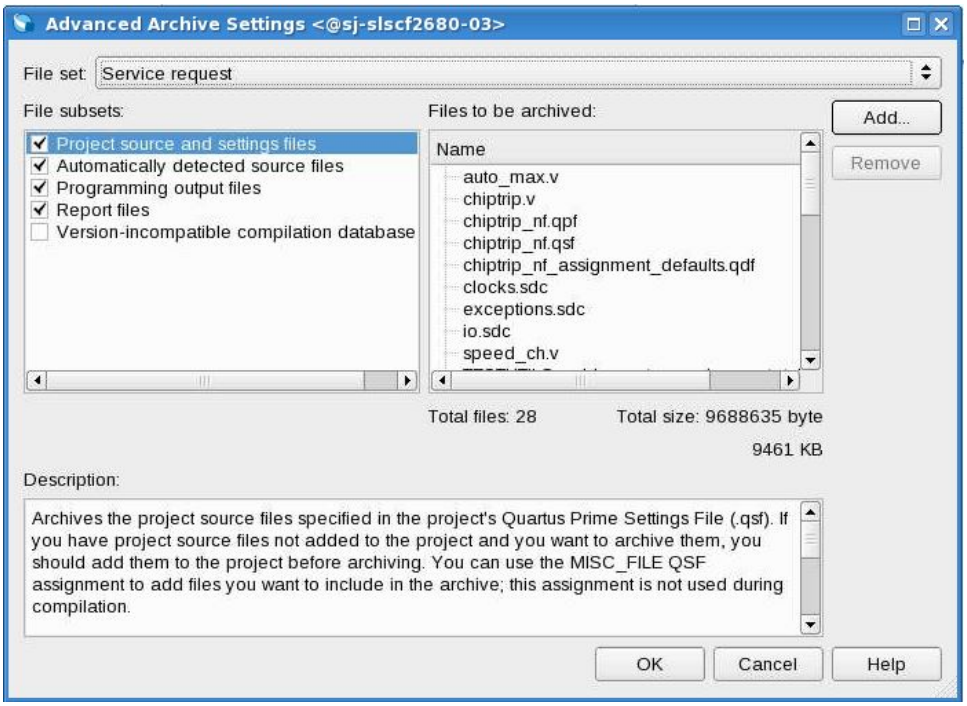
2.10.4.2 Archiving Projects for Service Requests

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

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

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

Figure 29. Archiving Project for Service Request



2.10.5 Using External Revision Control

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

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

2.10.5.1 Files to Include In External Revision Control

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

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

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

2.10.6 Migrating Projects Across Operating Systems

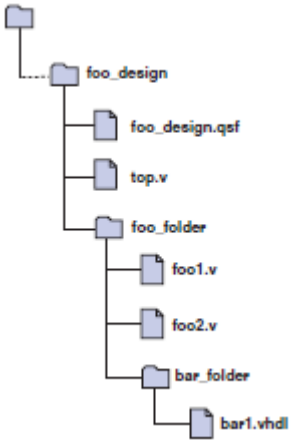
Consider the following cross-platform issues when moving your project from one operating system to another (for example, from Windows to Linux).

2.10.6.1 Migrating Design Files and Libraries

Consider file naming differences when migrating projects across operating systems.

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

Figure 30. All Inclusive Project Directory Structure

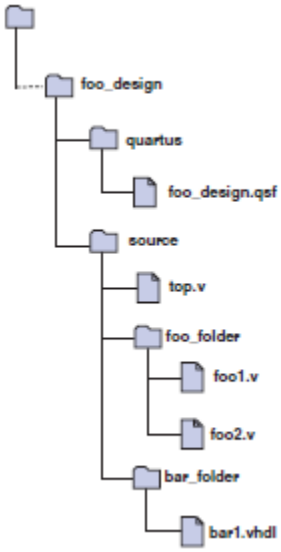


2.10.6.1.1 Use Relative Paths

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

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

Figure 31. Quartus Prime Project Directory Separate from Design Files



2.10.6.2 Design Library Migration Guidelines

The following guidelines apply to library migration across computing platforms:

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

2.11 Scripting API

Optionally use command-line executables or scripts to execute project commands, rather than using the GUI. The following commands are available for scripting project management.

2.11.1 Scripting Project Settings

Optionally use a Tcl script to specify settings and constraints, rather than using the GUI. This technique can be helpful if you have many settings and wish to track them in a single file or spreadsheet for iterative comparison. The **.qsf** supports only a limited subset of Tcl commands. Therefore, pass settings and constraints using a Tcl script:

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

2.11.2 Project Revision Commands

Use the following commands for scripting project revisions.

Create Revision Command on page 83
Set Current Revision Command on page 83
Get Project Revisions Command on page 83
Delete Revision Command on page 84

2.11.2.1 Create Revision Command

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

Option	Description
`based_on` (optional)	Specifies the revision name on which the new revision bases its settings.
`set_current` (optional)	Sets the new revision as the current revision.

2.11.2.2 Set Current Revision Command

The `-force` option enables you to open the revision that you specify under revision name and overwrite the compilation database if the database version is incompatible.

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

2.11.2.3 Get Project Revisions Command

```
get_project_revisions <project_name>
```

2.11.2.4 Delete Revision Command

```
delete_revision <revision name>
```

2.11.3 Project Archive Commands

Optionally use Tcl commands and the `quartus_sh` executable to create and manage archives of a Quartus project.

2.11.3.1 Creating a Project Archive

Use the following command to create a Quartus Prime archive:

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

You can specify the following other options:

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

2.11.3.2 Restoring an Archived Project

Use the following Tcl command to restore a Quartus project:

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

This example restores to a destination directory named "restored".

2.11.4 Project Database Commands

Use the following commands for managing Quartus compilation results:

Import and Export Version-Compatible Designs from the Design Flow on page 85

quartus_cdb Executables to Manage Version-Compatible Databases on page 85

2.11.4.1 Import and Export Version-Compatible Designs from the Design Flow

Optionally use Tcl commands to export and import a full design. The project must be open and the database must not be loaded before calling these commands. The provided snapshot will be loaded and unloaded by the Design Flow.

These commands require the `quartus_cdb` executable.

- To export a design's snapshot to a file:
  ```
  design::export_design -file <archive.qdb> -snapshot
  <snapshot_name>
      [-compatible]
  ```
- To import an exported design's snapshot into a project:
  ```
  design::import_design -file <archive.qdb> [-overwrite]
      [-timing_analysis_mode]
  ```

The `-compatible` option exports the database in a version-compatible format that can be imported into a newer version of the Quartus Prime software.

The `-overwrite` option removes existing project compilation databases before importing the archived `.qdb` file.

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

2.11.4.2 quartus_cdb Executables to Manage Version-Compatible Databases

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

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

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

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

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

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

2.11.5 Project Library Commands

Use the following commands to script project library changes.

Specify Project Libraries With SEARCH_PATH Assignment on page 86

Report Specified Project Libraries Commands on page 86

2.11.5.1 Specify Project Libraries With SEARCH_PATH Assignment

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

Use the following commands to script project library changes:

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

2.11.5.2 Report Specified Project Libraries Commands

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

Use the following commands to report specified project libraries:

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

2.12 Document Revision History

This document has the following revision history.

Table 24. Document Revision History

Date	Version	Changes
2017.05.08	17.0.0	• Added Project Tasks pane and update New Project Wizard. • Updated Compilation Dashboard image to show concurrent analysis. • Removed Smart Compilation option from Settings dialog screenshot. • Updated Qsys and IP Catalog screenshots for latest GUIs. • Added topic on Back-Annotate Assignments command.

continued...

Date	Version	Changes
		• Added Exporting a Design Partition topic. • Removed mentions to deprecated Incremental Compilation. • Added reference to Block-Level Design Flows.
2016.10.31	16.1.0	• Added references to compilation stages and snapshots. • Removed support for comparing revisions. • Added references to `.ip` file creation during Quartus Prime Pro Edition stand-alone IP generation. • Updated IP Core Generation Output files list and diagram. • Added Support for IP Core Encyption topic. • Rebranding for Intel
2016.05.03	16.0.0	• Removed statements about serial equivalence when using multiple processors. • Added the "Preserving Compilation Results" section. • Added the "Migrating Results Across Quartus Prime Software" section and its subsections for information about importing and exporting compilation results between different versions of Quartus Prime. • Added the "Project Database Commands" section and its subsections.
2016.02.09	15.1.1	• Clarified instructions for Generating a Combined Simulator Setup Script. • Clarified location of **Save project output files in specified directory** option.
2015.11.02	15.1.0	• Added Generating Version-Independent IP Simulation Scripts topic. • Added example IP simulation script templates for supported simulators. • Added Incorporating IP Simulation Scripts in Top-Level Scripts topic. • Added Troubleshooting IP Upgrade topic. • Updated IP Catalog and parameter editor descriptions for GUI changes. • Updated IP upgrade and migration steps for latest GUI changes. • Updated Generating IP Cores process for GUI changes. • Updated Files Generated for IP Cores and Qsys system description. • Removed references to devices and features not supported in version 15.1. • Changed instances of *Quartus II* to *Quartus Prime*.
2015.05.04	15.0.0	• Added description of design templates feature. • Updated screenshot for DSE II GUI. • Added qsys_script IP core instantiation information. • Described changes to generating and processing of instance and entity names. • Added description of upgrading IP cores at the command line. • Updated procedures for upgrading and migrating IP cores. • Gate level timing simulation supported only for Cyclone IV and Stratix IV devices.
2014.12.15	14.1.0	• Updated content for DSE II GUI and optimizations. • Added information about new **Assignments ➤ Settings ➤ IP Settings** that control frequency of synthesis file regeneration and automatic addtion of IP files to the project.

continued...

Date	Version	Changes
2014.08.18	14.0a10.0	• Added information about specifying parameters for IP cores targeting Arria 10 devices. • Added information about the latest IP output for version 14.0a10 targeting Arria 10 devices. • Added information about individual migration of IP cores to the latest devices. • Added information about editing existing IP variations.
2014.06.30	14.0.0	• Replaced MegaWizard Plug-In Manager information with IP Catalog. • Added standard information about upgrading IP cores. • Added standard installation and licensing information. • Removed outdated device support level information. IP core device support is now available in IP Catalog and parameter editor.
November 2013	13.1.0	• Conversion to DITA format
May 2013	13.0.0	• Overhaul for improved usability and updated information.
June 2012	12.0.0	• Removed survey link. • Updated information about `VERILOG_INCLUDE_FILE`.
November 2011	10.1.1	Template update.
December 2010	10.1.0	• Changed to new document template. • Removed Figure 4–1, Figure 4–6, Table 4–2. • Moved "Hiding Messages" to Help. • Removed references about the `set_user_option` command. • Removed Classic Timing Analyzer references.

Related Links

Altera Documentation Archive

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

3 Design Planning with the Quartus Prime Software

3.1 Design Planning with the Quartus Prime Software

Platform planning—the early feasibility analysis of physical constraints—is a fundamental early step in advanced FPGA design. FPGA device densities and complexities are increasing and designs often involve multiple designers. System architects must also resolve design issues when integrating design blocks. However, you can solve potential problems early in the design cycle by following the design planning considerations in this chapter.

Note: The BluePrint Platform Designer helps you to accurately plan constraints for design implementation. Use BluePrint to prototype interface implementations and rapidly define a legal device floorplan for Arria 10 devices.

Before reading the design planning guidelines discussed in this chapter, consider your design priorities. More device features, density, or performance requirements can increase system cost. Signal integrity and board issues can impact I/O pin locations. Power, timing performance, and area utilization all affect each other. Compilation time is affected when optimizing these priorities.

The Quartus Prime software optimizes designs for the best overall results; however, you can change the settings to better optimize one aspect of your design, such as power utilization. Certain tools or debugging options can lead to restrictions in your design flow. Your design priorities help you choose the tools, features, and methodologies to use for your design.

After you select a device family, to check if additional guidelines are available, refer to the design guidelines section of the appropriate device documentation.

Related Links

BluePrint Design Planning

3.2 Creating Design Specifications

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

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

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

Related Links

- Planning for On-Chip Debugging Tools on page 96
  Evaluate on-chip debugging tools early in your design process, Making changes to include debugging tools further in the design process is more time consumming and error prone.
- Using Qsys Pro and Standard Interfaces in System Design on page 90
  For improved reusability and ease of integration.

3.3 Selecting Intellectual Property Cores

Intel and its third-party intellectual property (IP) partners offer a large selection of standardized IP cores optimized for Intel devices. The IP you select often affects system design, especially if the FPGA interfaces with other devices in the system. Consider which I/O interfaces or other blocks in your system design are implemented using IP cores, and plan to incorporate these cores in your FPGA design.

The OpenCore Plus feature, which is available for many IP cores, allows you to program the FPGA to verify your design in the hardware before you purchase the IP license. The evaluation supports the following modes:

- Untethered—the design runs for a limited time.
- Tethered—the design requires an Intel serial JTAG cable connected between the JTAG port on your board and a host computer running the Quartus Prime Programmer for the duration of the hardware evaluation period.

Related Links

Intellectual Property
For descriptions of available IP cores.

3.4 Using Qsys Pro and Standard Interfaces in System Design

You can use the Quartus Prime Qsys Pro system integration tool to create your design with fast and easy system-level integration. With Qsys Pro, you can specify system components in a GUI and generate the required interconnect logic automatically, along with adapters for clock crossing and width differences.

Because system design tools change the design entry methodology, you must plan to start developing your design within the tool. Ensure all design blocks use appropriate standard interfaces from the beginning of the design cycle so that you do not need to make changes later.

Qsys Pro components use Avalon® standard interfaces for the physical connection of components, and you can connect any logical device (either on-chip or off-chip) that has an Avalon interface. The Avalon Memory-Mapped interface allows a component to use an address mapped read or write protocol that enables flexible topologies for connecting master components to any slave components. The Avalon Streaming interface enables point-to-point connections between streaming components that send and receive data using a high-speed, unidirectional system interconnect between source and sink ports.

In addition to enabling the use of a system integration tool such as Qsys Pro, using standard interfaces ensures compatibility between design blocks from different design teams or vendors. Standard interfaces simplify the interface logic to each design block and enable individual team members to test their individual design blocks against the specification for the interface protocol to ease system integration.

Related Links

- System Design with Qsys Pro
  For more information about using Qsys Pro to improve your productivity.
- SOPC Builder User Guide
  For more information about SOPC Builder.

3.5 Device Selection

The device you choose affects board specification and layout. Use the following guidelines for selecting a device.

Choose the device family that best suits your design requirements. Families differ in cost, performance, logic and memory density, I/O density, power utilization, and packaging. You must also consider feature requirements, such as I/O standards support, high-speed transceivers, global or regional clock networks, and the number of phase-locked loops (PLLs) available in the device.

Each device family has complete documentation, including a data sheet, which documents device features in detail. You can also see a summary of the resources for each device in the **Device** dialog box in the Quartus Prime software.

Carefully study the device density requirements for your design. Devices with more logic resources and higher I/O counts can implement larger and more complex designs, but at a higher cost. Smaller devices use lower static power. Select a device larger than what your design requires if you want to add more logic later in the design cycle to upgrade or expand your design, and reserve logic and memory for on-chip debugging. Consider requirements for types of dedicated logic blocks, such as memory blocks of different sizes, or digital signal processing (DSP) blocks to implement certain arithmetic functions.

If you have older designs that target an Intel device, you can use their resources as an estimate for your design. Compile existing designs in the Quartus Prime software with the **Auto device selected by the Fitter** option in the **Settings** dialog box. Review the resource utilization to learn which device density fits your design. Consider coding style, device architecture, and the optimization options used in the Quartus Prime software, which can significantly affect the resource utilization and timing performance of your design.

Related Links

- Planning for On-Chip Debugging Tools on page 96
  For information about on-chip debugging.
- Altera Product Selector
  You can refer to the Altera website to help you choose your device.
- Selector Guides
  You can review important features of each device family in the refer to the Altera website.
- Devices and Adapters

For a list of device selection guides.

- IP and Megafunctions
  For information on how to obtain resource utilization estimates for certain configurations of Intel's FPGA IP, refer to the user guides for Intel FPGA megafunctions and IP MegaCores on the literature page of the Altera website.

3.5.1 Device Migration Planning

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

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

3.6 Development Kit Selection

In addition to specifying the device you want to target for compilation, you can also specify a target board or a development kit for your design.When you select a development kit, the Quartus Prime software provides a kit reference design, and creates pin assignments for the kit.

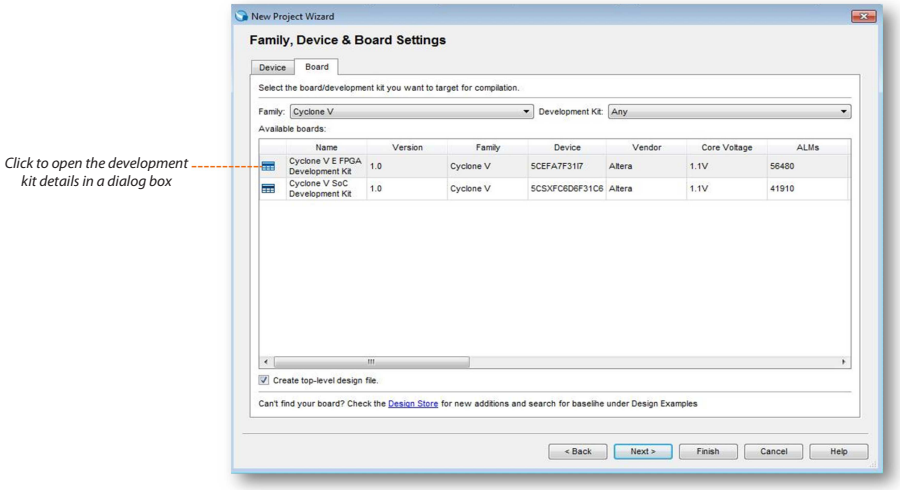
You can select a development kit for your new Quartus Prime project from the **New Project Wizard**, or for an existing project by clicking **Assignments ➤ Device**.

3.6.1 Specifying a Development Kit for a New Project

Follow the steps below to select a development kit for a new Quartus Prime project:

1. To open the **New Project Wizard**, click **File ➤ New Project Wizard**.
2. Click the **Board** tab from **Family, Device & Board Settings** page.
3. Select the **Family** and/or **Development Kit** list to narrow your board search. The **Available boards** table lists all the available boards for the selected **Family** and **Development Kit** type.
4. To view the development kit details for each of the listed boards, click the icons to the left of the boards in the **Available boards** table. The **Development Kit Details** dialog box appears, displaying all the board details.
5. Select the desired board from the **Available boards** table.
6. To set the selected board design as top-level entity, click the **Create top-level design file** checkbox. This option automatically sets up the pin assignments for the selected board. If you choose to uncheck this option, the Quartus Prime software creates the design for the board and stores the design in `<current_project_dir>/devkits/<design_name>`.
7. Click **Finish**.

Figure 32. Selecting the Desired Board from New Project Wizard



Note: If you are unable to find the board you are looking for in the **Available Boards** table, click **Design Store** link at the bottom of the page. This link takes you to the Altera development store from where you can purchase development kits and download baseline design examples.

Related Links

Design Store

3.6.2 Specifying a Development Kit for an Existing Project

Follow the steps below to select a development kit for your existing Quartus Prime project:

1. To open your existing project, click **File ➤ Open Project**.
2. To open the **Device Setting Dialog Box**, click **Assignments ➤ Device**.
3. Select the desired development kit from the **Board** tab and click **OK**.
4. If there are existing pin assignments in your current project, a message box appears, prompting to remove all location assignments. Click **Yes** to remove the **Location** and **I/O Standard** pin assignments. The Quartus Prime software creates the kit's baseline design and stores the design in `<current_project_dir>/devkits/<design_name>`. To retain all your existing pin assignments, click **No**.

Note: Repeat the above steps to change the development kit of an existing project.

3.6.3 Setting Pin Assignments

The `<design_name>` folder contains the `platform_setup.tcl file` that stores all the pin assignments and the baseline example designs for the board. In addition, the Quartus Prime software creates a `.qdf` file in the `<current_project_dir>` folder, which stores all the default values for the pin assignments.

To manually set up the pin assignments:

1. Click **View ➤ Tcl Console**.
2. At the Tcl console command prompt, type the command:

```
source <current_project_dir>/devkits/<design_name>/platform_setup.tcl
```

3. At the Tcl console command prompt, type the command:

```
setup_project
```

This command populates all assignments available in the `setup_platform.tcl` file to your `.qsf` file.

3.7 Planning for Device Programming or Configuration

System planning includes determining what companion devices, if any, your system requires. Your board layout also depends on the type of programming or configuration method you plan to use for programmable devices. Many programming options require a JTAG interface to connect to the devices, so you might have to set up a JTAG chain on the board. Additionally, the Quartus Prime software uses the settings for the configuration scheme, configuration device, and configuration device voltage to enable the appropriate dual purpose pins as regular I/O pins after you complete configuration. The Quartus Prime software performs voltage compatibility checks of those pins during compilation of your design. Use the **Configuration** tab of the **Device and Pin Options** dialog box to select your configuration scheme.

The device family documentation describes the configuration options available for a device family. For information about programming CPLD devices, refer to your device documentation.

Related Links

Configuration Handbook
For more details about configuration options.

3.8 Estimating Power

You can use the Quartus Prime power estimation and analysis tools to provide information to PCB board and system designers. Power consumption in FPGA devices depends on the design logic, which can make planning difficult. You can estimate power before you create any source code, or when you have a preliminary version of the design source code, and then perform the most accurate analysis with the Power Analyzer when you complete your design.

You must accurately estimate device power consumption to develop an appropriate power budget and to design the power supplies, voltage regulators, heat sink, and cooling system. Power estimation and analysis helps you satisfy two important planning requirements:

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

The Early Power Estimator (EPE) spreadsheet allows you to estimate power utilization for your design.

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

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

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

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

Estimating power consumption early in the design cycle allows planning of power budgets and avoids unexpected results when designing the PCB.

When you complete your design, perform a complete power analysis to check the power consumption more accurately. The Power Analyzer tool in the Quartus Prime software provides an accurate estimation of power, ensuring that thermal and supply limitations are met.

Related Links

- Power Analysis
  For more information about power estimation and analysis.
- Early Power Estimator and Power Analyzer
  The EPE spreadsheets for each supported device family are available on the Altera website.

3.9 Selecting Third-Party EDA Tools

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

3.9.1 Synthesis Tool

You can use supported standard third-party EDA synthesis tools to synthesize your Verilog HDL or VHDL design, and then compile the resulting output netlist file in the Quartus Prime software.

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

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

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

3.9.2 Simulation Tool

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

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

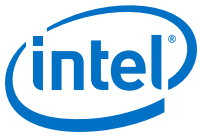
3.9.3 Formal Verification Tools

Consider whether the Quartus Prime software supports the formal verification tool that you want to use, and whether the flow impacts your design and compilation stages of your design.

Using a formal verification tool can impact performance results because performing formal verification requires turning off certain logic optimizations, such as register retiming, and forces you to preserve hierarchy blocks, which can restrict optimization. Formal verification treats memory blocks as black boxes. Therefore, you must keep memory in a separate hierarchy block so other logic does not get incorporated into the black box for verification. If formal verification is important to your design, plan for limitations and restrictions at the beginning of the design cycle rather than make changes later.

3.10 Planning for On-Chip Debugging Tools

Evaluate on-chip debugging tools early in your design process, Making changes to include debugging tools further in the design process is more time consumming and error prone.



In-system debugging tools offer different advantages and trade-offs. A particular debugging tool may work better for different systems and designers. Consider the following debugging requirements when you plan your design:

- JTAG connections—required to perform in-system debugging with JTAG tools. Plan your system and board with JTAG ports that are available for debugging.
- Additional logic resources (ALR)—required to implement JTAG hub logic. If you set up the appropriate tool early in your design cycle, you can include these device resources in your early resource estimations to ensure that you do not overload the device with logic.
- Reserve device memory—required if your tool uses device memory to capture data during system operation. To ensure that you have enough memory resources to take advantage of this debugging technique, consider reserving device memory to use during debugging.
- Reserve I/O pins—required if you use the Logic Analyzer Interface (LAI), which require I/O pins for debugging. If you reserve I/O pins for debugging, you do not have to later change your design or board. The LAI can multiplex signals with design I/O pins if required. Ensure that your board supports a debugging mode, in which debugging signals do not affect system operation.
- Instantiate an IP core in your HDL code—required if your debugging tool uses an Intel FPGA IP core.
- Instantiate the Signal Tap Logic Analyzer IP core—required if you want to manually connect the Signal Tap Logic Analyzer to nodes in your design and ensure that the tapped node names do not change during synthesis.

Table 25. Factors to Consider When Using Debugging Tools During Design Planning Stages

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

Related Links

- System Debugging Tools Overview
  In *Quartus Prime Pro Edition Handbook Volume 3*
- Design Debugging Using the Signal Tap Logic Analyzer
  In *Quartus Prime Pro Edition Handbook Volume 3*

3.11 Design Practices and HDL Coding Styles

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

3.11.1 Design Recommendations

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

In a synchronous design, a clock signal triggers all events. When you meet all register timing requirements, a synchronous design behaves in a predictable and reliable manner for all process, voltage, and temperature (PVT) conditions. You can easily target synchronous designs to different device families or speed grades.

Clock signals have a large effect on the timing accuracy, performance, and reliability of your design. Problems with clock signals can cause functional and timing problems in your design. Use dedicated clock pins and clock routing for best results, and if you have PLLs in your target device, use the PLLs for clock inversion, multiplication, and division. For clock multiplexing and gating, use the dedicated clock control block or PLL clock switchover feature instead of combinational logic, if these features are available in your device. If you must use internally-generated clock signals, register the output of any combinational logic used as a clock signal to reduce glitches.

Consider the architecture of the device you choose so that you can use specific features in your design. For example, the control signals should use the dedicated control signals in the device architecture. Sometimes, you might need to limit the number of different control signals used in your design to achieve the best results.

Related Links

- Recommended Design Practices on page 155
  This chapter provides design recommendations for Intel FPGA devices.
- www.sunburst-design.com/papers
  You can also refer to industry papers for more information about multiple clock design. For a good analysis, refer to *Synthesis and Scripting Techniques for Designing Multi-Asynchronous Clock Designs*

3.11.2 Recommended HDL Coding Styles

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

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

Related Links

Recommended HDL Coding Styles on page 102
This chapter provides Hardware Description Language (HDL) coding style recommendations to ensure optimal synthesis results when targeting Intel FPGA devices.

3.11.3 Managing Metastability

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

Designers commonly use a synchronization chain to minimize the occurrence of metastable events. Ensure that your design accounts for synchronization between any asynchronous clock domains. Consider using a synchronizer chain of more than two registers for high-frequency clocks and frequently-toggling data signals to reduce the chance of a metastability failure.

You can use the Quartus Prime software to analyze the average mean time between failures (MTBF) due to metastability when a design synchronizes asynchronous signals, and optimize your design to improve the metastability MTBF. The MTBF due to metastability is an estimate of the average time between instances when metastability could cause a design failure. A high MTBF (such as hundreds or thousands of years between metastability failures) indicates a more robust design. Determine an acceptable target MTBF given the context of your entire system and the fact that MTBF calculations are statistical estimates.

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

Related Links

Managing Metastability with the Quartus Prime Software on page 952
For information about metastability analysis, reporting, and optimization features in the Quartus Prime software

3.12 Running Fast Synthesis

You save time when you find design issues early in the design cycle rather than in the final timing closure stages. When the first version of the design source code is complete, you might want to perform a quick compilation to create a kind of silicon virtual prototype (SVP) that you can use to perform timing analysis.

If you synthesize with the Quartus Prime software, you can choose to perform a **Fast** synthesis, which reduces the compilation time, but may give reduced quality of results.

If you design individual design blocks or partitions separately, you can use the Fast synthesis and early timing estimate features as you develop your design. Any issues highlighted in the lower-level design blocks are communicated to the system architect. Resolving these issues might require allocating additional device resources to the individual partition, or changing the timing budget of the partition.

Related Links

Synthesis Effort logic option
For more information about Fast synthesis, refer to Quartus Prime Help.

3.13 Document Revision History

Table 26. Document Revision History

Date	Version	Changes
2017.05.08	17.0.0	• Removed mentions to Integrated Synthesis.
2016.10.31	16.1.0	• Implemented Intel rebranding.
2016.05.03	16.0.0	Added information about Development Kit selection.
2015.11.02	15.1.0	• Added references to BluePrint Design Planning chapter. • Changed instances of *Quartus II* to *Quartus Prime*.
2015.05.04	15.0.0	Remove support for Early Timing Estimate feature.
2014.06.30	14.0.0	Updated document format.
November 2013	13.1.0	Removed HardCopy device information.
November, 2012	12.1.0	Update for changes to early pin planning feature
June 2012	12.0.0	Editorial update.
November 2011	11.0.1	Template update.
May 2011	11.0.0	• Added link to System Design with Qsys in "Creating Design Specifications" on page 1–2 • Updated "Simultaneous Switching Noise Analysis" on page 1–8 • Updated "Planning for On-Chip Debugging Tools" on page 1–10 • Removed information from "Planning Design Partitions and Floorplan Location Assignments" on page 1–15
December 2010	10.1.0	• Changed to new document template • Updated "System Design and Standard Interfaces" on page 1–3 to include information about the Qsys system integration tool • Added link to the Altera Product Selector in "Device Selection" on page 1–3 • Converted information into new table (Table 1–1) in "Planning for On-Chip Debugging Options" on page 1–10 • Simplified description of incremental compilation usages in "Incremental Compilation with Design Partitions" on page 1–14 • Added information about the Rapid Recompile option in "Flat Compilation Flow with No Design Partitions" on page 1–14 • Removed details and linked to Quartus Prime Help in "Fast Synthesis and Early Timing Estimation" on page 1–16
July 2010	10.0.0	• Added new section "System Design" on page 1–3 • Removed details about debugging tools from "Planning for On-Chip Debugging Options" on page 1–10 and referred to other handbook chapters for more information • Updated information on recommended design flows in "Incremental Compilation with Design Partitions" on page 1–14 and removed "Single-Project Versus Multiple-Project Incremental Flows" heading • Merged the "Planning Design Partitions" section with the "Creating a Design Floorplan" section. Changed heading title to "Planning Design Partitions and Floorplan Location Assignments" on page 1–15

continued...

Date	Version	Changes
		• Removed "Creating a Design Floorplan" section • Removed "Referenced Documents" section • Minor updates throughout chapter
November 2009	9.1.0	• Added details to "Creating Design Specifications" on page 1–2 • Added details to "Intellectual Property Selection" on page 1–2 • Updated information on "Device Selection" on page 1–3 • Added reference to "Device Migration Planning" on page 1–4 • Removed information from "Planning for Device Programming or Configuration" on page 1–4 • Added details to "Early Power Estimation" on page 1–5 • Updated information on "Early Pin Planning and I/O Analysis" on page 1–6 • Updated information on "Creating a Top-Level Design File for I/O Analysis" on page 1–8 • Added new "Simultaneous Switching Noise Analysis" section • Updated information on "Synthesis Tools" on page 1–9 • Updated information on "Simulation Tools" on page 1–9 • Updated information on "Planning for On-Chip Debugging Options" on page 1–10 • Added new "Managing Metastability" section • Changed heading title "Top-Down Versus Bottom-Up Incremental Flows" to "Single-Project Versus Multiple-Project Incremental Flows" • Updated information on "Creating a Design Floorplan" on page 1–18 • Removed information from "Fast Synthesis and Early Timing Estimation" on page 1–18
March 2009	9.0.0	• No change to content
November 2008	8.1.0	• Changed to 8-1/2 x 11 page size. No change to content.
May 2008	8.0.0	• Organization changes • Added "Creating Design Specifications" section • Added reference to new details in the In-System Design Debugging section of volume 3 • Added more details to the "Design Practices and HDL Coding Styles" section • Added references to the new Best Practices for Incremental Compilation and Floorplan Assignments chapter • Added reference to the Quartus Prime Language Templates

Related Links

Altera Documentation Archive

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

4 Recommended HDL Coding Styles

This chapter provides Hardware Description Language (HDL) coding style recommendations to ensure optimal synthesis results when targeting Intel FPGA devices.

HDL coding styles have a significant effect on the quality of results for programmable logic designs. Synthesis tools optimize HDL code for both logic utilization and performance; however, synthesis tools cannot interpret the intent of your design. Therefore, the most effective optimizations require conformance to recommended coding styles. Refer to the Altera website for design examples that conform to these standards.

Note: For style recommendations, options, or HDL attributes specific to your synthesis tool (including other Quartus software products and other EDA tools), refer to the synthesis tool vendor's documentation.

Related Links

- Advanced Synthesis Cookbook
- Design Examples
- Reference Designs

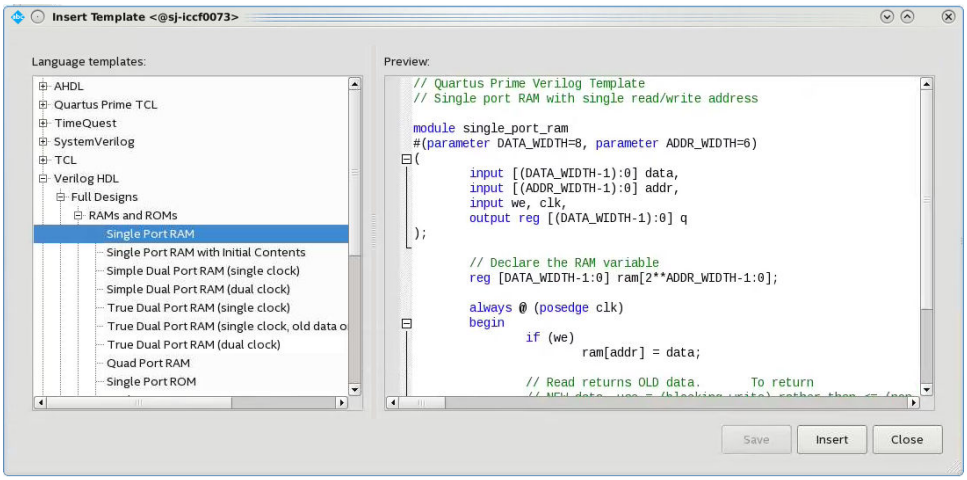
4.1 Using Provided HDL Templates

The Quartus Prime software provides templates for Verilog HDL, SystemVerilog, and VHDL templates to start your HDL designs. Many of the HDL examples in this document correspond with the **Full Designs** examples in the **Quartus Prime Templates**. You can insert HDL code into your own design using the templates or examples.

4.1.1 Inserting HDL Code from a Provided Template

1. Click **File ➤ New**.
2. In the **New** dialog box, select the type of design file corresponding to the type of HDL you want to use: **SystemVerilog HDL File**, **VHDL File**, or **Verilog HDL File**; and click **OK**. A text editor tab with a blank file opens.
3. Right-click the blank file, and click **Insert Template...**.
4. In the **Insert Template** dialog box, expand the section corresponding to the appropriate HDL, then expand the **Full Designs** section.
5. Select a template. The HDL appears in the **Preview** pane.
6. To paste the HDL design into the blank Verilog or VHDL file you created, click **Insert**.
7. Close the **Insert Template** dialog box by clicking **Close**.

Figure 33. Inserting a RAM Template



Note: Use the Quartus Prime Text Editor to modify the HDL design or save the template as an HDL file to edit in your preferred text editor.

4.2 Instantiating IP Cores in HDL

Intel provides parameterizable IP cores that are optimized for Intel FPGA device architectures. Using IP cores instead of coding your own logic saves valuable design time.

Additionally, the Intel-provided IP cores offer more efficient logic synthesis and device implementation. Scale the IP core's size and specify various options by setting parameters. To instantiate the IP core directly in your HDL file code, invoke the IP core name and define its parameters as you would do for any other module, component, or subdesign. Alternatively, you can use the IP Catalog (**Tools ➤ IP Catalog**) and parameter editor GUI to simplify customization of your IP core variation. You can infer or instantiate IP cores that optimize device architecture features, for example:

- Transceivers
- LVDS drivers
- Memory and DSP blocks
- Phase-locked loops (PLLs)
- Double-data rate input/output (DDIO) circuitry

For some types of logic functions, such as memories and DSP functions, you can infer device-specific dedicated architecture blocks instead of instantiating an IP core. Quartus Prime synthesis recognizes certain HDL code structures and automatically infers the appropriate IP core or map directly to device atoms.

Related Links

Intel FPGA IP Core Literature

4.3 Inferring Multipliers and DSP Functions

The following sections describe how to infer multiplier and DSP functions from generic HDL code, and, if applicable, how to target the dedicated DSP block architecture in Intel FPGA devices.

Related Links

DSP Solutions Center

4.3.1 Inferring Multipliers

To infer multiplier functions, synthesis tools detect multiplier logic and implement this in Intel FPGA IP cores, or map the logic directly to device atoms.

For devices with DSP blocks, Quartus Prime synthesis can implement the function in a DSP block instead of logic, depending on device utilization. The Quartus Prime fitter can also place input and output registers in DSP blocks (that is, perform register packing) to improve performance and area utilization.

The following Verilog HDL and VHDL code examples show that synthesis tools can infer signed and unsigned multipliers as IP cores or DSP block atoms. Each example fits into one DSP block element. In addition, when register packing occurs, no extra logic cells for registers are required.

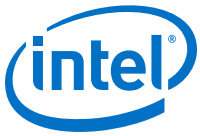
Example 6. Verilog HDL Unsigned Multiplier

```
module unsigned_mult (out, a, b);
    output [15:0] out;
    input [7:0] a;
    input [7:0] b;
    assign out = a * b;
endmodule
```

Note: The `signed` declaration in Verilog HDL is a feature of the Verilog 2001 Standard.

Example 7. Verilog HDL Signed Multiplier with Input and Output Registers (Pipelining = 2)

```
module signed_mult (out, clk, a, b);
   output [15:0] out;
   input clk;
   input signed [7:0] a;
   input signed [7:0] b;

   reg signed [7:0] a_reg;
   reg signed [7:0] b_reg;
   reg signed [15:0] out;
   wire signed [15:0] mult_out;

   assign mult_out = a_reg * b_reg;

   always @ (posedge clk)
   begin
      a_reg <= a;
      b_reg <= b;
      out <= mult_out;
   end
endmodule
```



Example 8. VHDL Unsigned Multiplier with Input and Output Registers (Pipelining = 2)

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

ENTITY unsigned_mult IS
   PORT (
      a: IN UNSIGNED (7 DOWNTO 0);
      b: IN UNSIGNED (7 DOWNTO 0);
      clk: IN STD_LOGIC;
      aclr: IN STD_LOGIC;
      result: OUT UNSIGNED (15 DOWNTO 0)
   );
END unsigned_mult;

ARCHITECTURE rtl OF unsigned_mult IS
   SIGNAL a_reg, b_reg: UNSIGNED (7 DOWNTO 0);
BEGIN
   PROCESS (clk, aclr)
   BEGIN
      IF (aclr ='1') THEN
         a_reg <= (OTHERS => '0');
         b_reg <= (OTHERS => '0');
         result <= (OTHERS => '0');
      ELSIF (rising_edge(clk)) THEN
         a_reg <= a;
         b_reg <= b;
         result <= a_reg * b_reg;
      END IF;
   END PROCESS;
END rtl;
```

Example 9. VHDL Signed Multiplier

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

ENTITY signed_mult IS
   PORT (
      a: IN SIGNED (7 DOWNTO 0);
      b: IN SIGNED (7 DOWNTO 0);
      result: OUT SIGNED (15 DOWNTO 0)
   );
END signed_mult;

ARCHITECTURE rtl OF signed_mult IS
BEGIN
   result <= a * b;
END rtl;
```

4.3.2 Inferring Multiply-Accumulator and Multiply-Adder Functions

Synthesis tools detect multiply-accumulator or multiply-adder functions, and either implement them as Intel FPGA IP cores or map them directly to device atoms. During placement and routing, the Quartus Prime software places multiply-accumulator and multiply-adder functions in DSP blocks.

Note: Synthesis tools infer multiply-accumulator and multiply-adder functions only if the Intel device family has dedicated DSP blocks that support these functions.

A simple multiply-accumulator consists of a multiplier feeding an addition operator. The addition operator feeds a set of registers that then feeds the second input to the addition operator. A simple multiply-adder consists of two to four multipliers feeding one or two levels of addition, subtraction, or addition/subtraction operators. Addition is always the second-level operator, if it is used. In addition to the multiply-accumulator and multiply-adder, the Quartus Prime Fitter also places input and output registers into the DSP blocks to pack registers and improve performance and area utilization.

Some device families offer additional advanced multiply-adder and accumulator functions, such as complex multiplication, input shift register, or larger multiplications.

The Verilog HDL and VHDL code samples infer multiply-accumulator and multiply-adder functions with input, output, and pipeline registers, as well as an optional asynchronous `clear` signal. Using the three sets of registers provides the best performance through the function, with a latency of three. To reduce latency, remove the registers in your design.

Note: To obtain high performance in DSP designs, use register pipelining and avoid unregistered DSP functions.

Example 10. Verilog HDL Multiply-Accumulator

```
module sum_of_four_multiply_accumulate
   #(parameter INPUT_WIDTH=18, parameter OUTPUT_WIDTH=44)
   (
      input clk, ena,
      input [INPUT_WIDTH-1:0] dataa, datab, datac, datad,
      input [INPUT_WIDTH-1:0] datae, dataf, datag, datah,
      output reg [OUTPUT_WIDTH-1:0] dataout
   );
   // Each product can be up to 2*INPUT_WIDTH bits wide.
   // The sum of four of these products can be up to 2 bits wider.
   wire [2*INPUT_WIDTH+1:0] mult_sum;

   // Store the results of the operations on the current inputs
   assign mult_sum = (dataa * datab + datac * datad) + \
                     (datae * dataf + datag * datah);

   // Store the value of the accumulation
   always @ (posedge clk)
   begin
      if (ena == 1)
         begin
            dataout <= dataout + mult_sum;
         end
   end
endmodule
```

Related Links

- DSP Design Examples
- AN639: Inferring Stratix V DSP Blocks for FIR Filtering

4.4 Inferring Memory Functions from HDL Code

The following coding recommendations provide portable examples of generic HDL code targeting dedicated Intel FPGA memory IP cores. However, if you want to use some of the advanced memory features in Intel FPGA devices, consider using the IP core directly so that you can customize the ports and parameters easily.

You can also use the Quartus Prime templates provided in the Quartus Prime software as a starting point. Most of these designs can also be found on the Design Examples page on the Altera website.

Table 27. Intel Memory HDL Language Templates

Language	Full Design Name
VHDL	Single-Port RAM Single-Port RAM with Initial Contents Simple Dual-Port RAM (single clock) Simple Dual-Port RAM (dual clock) True Dual-Port RAM (single clock) True Dual-Port RAM (dual clock) Mixed-Width RAM Mixed-Width True Dual-Port RAM Byte-Enabled Simple Dual-Port RAM Byte-Enabled True Dual-Port RAM Single-Port ROM Dual-Port ROM
Verilog HDL	Single-Port RAM Single-Port RAM with Initial Contents Simple Dual-Port RAM (single clock) Simple Dual-Port RAM (dual clock) True Dual-Port RAM (single clock) True Dual-Port RAM (dual clock) Single-Port ROM Dual-Port ROM
SystemVerilog	Mixed-Width Port RAM Mixed-Width True Dual-Port RAM Mixed-Width True Dual-Port RAM (new data on same port read during write) Byte-Enabled Simple Dual Port RAM Byte-Enabled True Dual-Port RAM

Related Links

- Instantiating IP Cores in HDL
  In *Introduction to Intel FPGA IP Cores*
- Design Examples
- Embedded Memory Blocks in Arria 10 Devices
  In *Intel Arria 10 Core Fabric and General Purpose I/Os Handbook*

4.4.1 Inferring RAM functions from HDL Code

To infer RAM functions, synthesis tools recognize certain types of HDL code and map the detected code to technology-specific implementations. For device families that have dedicated RAM blocks, the Quartus Prime software uses an Intel FPGA IP core to target the device memory architecture.

Synthesis tools typically consider all signals and variables that have a multi-dimensional array type and then create a RAM block, if applicable. This is based on the way the signals or variables are assigned or referenced in the HDL source description.

Standard synthesis tools recognize single-port and simple dual-port (one read port and one write port) RAM blocks. Some synthesis tools (such as the Quartus Prime software) also recognize true dual-port (two read ports and two write ports) RAM blocks that map to the memory blocks in certain Intel FPGA devices.

Some tools (such as the Quartus Prime software) also infer memory blocks for array variables and signals that are referenced (read/written) by two indexes, to recognize mixed-width and byte-enabled RAMs for certain coding styles.

Note: If your design contains a RAM block that your synthesis tool does not recognize and infer, the design might require a large amount of system memory that can potentially cause compilation problems.

4.4.1.1 Use Synchronous Memory Blocks

Memory blocks in Intel FPGA are synchronous. Therefore, RAM designs must be synchronous to map directly into dedicated memory blocks. For these devices, Quartus Prime synthesis implements asynchronous memory logic in regular logic cells.

Synchronous memory offers several advantages over asynchronous memory, including higher frequencies and thus higher memory bandwidth, increased reliability, and less standby power. To convert asynchronous memory, move registers from the datapath into the memory block.

A memory block is synchronous if it has one of the following read behaviors:

- Memory read occurs in a Verilog HDL `always` block with a `clock` signal or a VHDL clocked process. The recommended coding style for synchronous memories is to create your design with a registered read output.
- Memory read occurs outside a clocked block, but there is a synchronous read address (that is, the address used in the read statement is registered). Synthesis does not always infer this logic as a memory block, or may require external bypass logic, depending on the target device architecture. Avoid this coding style for synchronous memories.

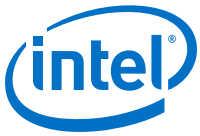
Note: The synchronous memory structures in Intel FPGA devices can differ from the structures in other vendors' devices. For best results, match your design to the target device architecture.

This chapter provides coding recommendations for various memory types. All of the examples in this document are synchronous to ensure that they can be directly mapped into the dedicated memory architecture available in Intel FPGAs.

4.4.1.2 Avoid Unsupported Reset and Control Conditions

To ensure correct implementation of HDL code in the target device architecture, avoid unsupported reset conditions or other control logic that does not exist in the device architecture.

The RAM contents of Intel FPGA memory blocks cannot be cleared with a `reset` signal during device operation. If your HDL code describes a RAM with a `reset` signal for the RAM contents, the logic is implemented in regular logic cells instead of a memory block. Do not place RAM read or write operations in an `always` block or `process` block with a `reset` signal. To specify memory contents, initialize the memory or write the data to the RAM during device operation.



In addition to reset signals, other control logic can prevent synthesis from inferring memory logic as a memory block. For example, if you use a clock enable on the read address registers, you can alter the output latch of the RAM, resulting in the synthesized RAM result not matching the HDL description. Use the address stall feature as a read address clock enable to avoid this limitation. Check the documentation for your FPGA device to ensure that your code matches the hardware available in the device.

Example 11. Verilog RAM with Reset Signal that Clears RAM Contents: Not Supported in Device Architecture

```
module clear_ram
(
    input clock, reset, we,
    input [7:0] data_in,
    input [4:0] address,
    output reg [7:0] data_out
);

    reg [7:0] mem [0:31];
    integer i;

    always @ (posedge clock or posedge reset)
    begin
        if (reset == 1'b1)
            mem[address] <= 0;
        else if (we == 1'b1)
            mem[address] <= data_in;

        data_out <= mem[address];
    end
endmodule
```

Example 12. Verilog RAM with Reset Signal that Affects RAM: Not Supported in Device Architecture

```
module bad_reset
(
    input clock,
    input reset,
    input we,
    input [7:0] data_in,
    input [4:0] address,
    output reg [7:0] data_out,
    input d,
    output reg q
);

    reg [7:0] mem [0:31];
    integer i;

    always @ (posedge clock or posedge reset)
    begin
        if (reset == 1'b1)
            q <= 0;
        else
        begin
            if (we == 1'b1)
                mem[address] <= data_in;

            data_out <= mem[address];
            q <= d;
```

```
        end
    end
endmodule
```

Related Links

Specifying Initial Memory Contents at Power-Up on page 122

4.4.1.3 Check Read-During-Write Behavior

Ensure the read-during-write behavior of the memory block described in your HDL design is consistent with your target device architecture.

Your HDL source code specifies the memory behavior when you read and write from the same memory address in the same clock cycle. The read returns either the old data at the address, or the new data written to the address. This is referred to as the read-during-write behavior of the memory block. Intel FPGA memory blocks have different read-during-write behavior depending on the target device family, memory mode, and block type.

Synthesis tools preserve the functionality described in your source code. Therefore, if your source code specifies unsupported read-during-write behavior for the RAM blocks, the Quartus Prime software implements the logic in regular logic cells as opposed to the dedicated RAM hardware.

Example 13. Continuous read in HDL code

One common problem occurs when there is a continuous read in the HDL code, as in the following examples. Avoid using these coding styles:

```
//Verilog HDL concurrent signal assignment
assign q = ram[raddr_reg];
```

```
-- VHDL concurrent signal assignment
q <= ram(raddr_reg);
```

This type of HDL implies that when a write operation takes place, the read immediately reflects the new data at the address independent of the read clock, which is the behavior of asynchronous memory blocks. Synthesis cannot directly map this behavior to a synchronous memory block. If the write clock and read clock are the same, synthesis can infer memory blocks and add extra bypass logic so that the device behavior matches the HDL behavior. If the write and read clocks are different, synthesis cannot reliably add bypass logic, so it implements the logic in regular logic cells instead of dedicated RAM blocks. The examples in the following sections discuss some of these differences for read-during-write conditions.

In addition, the MLAB memories in certain device logic array blocks (LABs) does not easily support old data or new data behavior for a read-during-write in the dedicated device architecture. Implementing the extra logic to support this behavior significantly reduces timing performance through the memory.

Note: For best performance in MLAB memories, ensure that your design does not depend on the read data during a write operation.

In many synthesis tools, you can declare that the read-during-write behavior is not important to your design (for example, if you never read from the same address to which you write in the same clock cycle). In Quartus Prime Pro Edition synthesis, set the synthesis attribute `ramstyle` to `no_rw_check` to allow Quartus Prime software to define the read-during-write behavior of a RAM, rather than use the behavior specified by your HDL code. This attribute can prevent the synthesis tool from using extra logic to implement the memory block, or can allow memory inference when it would otherwise be impossible.

4.4.1.4 Controlling RAM Inference and Implementation

Quartus Prime synthesis provides options to control RAM inference and implementation for Intel FPGA devices with synchronous memory blocks. Synthesis tools usually do not infer small RAM blocks because implementing small RAM blocks is more efficient if using the registers in regular logic.

To direct the Quartus Prime software to infer RAM blocks globally for all sizes, enable the **Allow Any RAM Size for Recognition** option in the **Advanced Analysis & Synthesis Settings** dialog box.

Alternatively, use the `ramstyle` RTL attribute to specify how an inferred RAM is implemented, including the type of memory block or the use of regular logic instead of a dedicated memory block. Quartus Prime synthesis does not map inferred memory into MLABs unless the HDL code specifies the appropriate `ramstyle` attribute, although the Fitter may map some memories to MLABs.

Set the `ramstyle` attribute in the RTL or in the `.qsf` file.

```
(* ramstyle = "mlab" *) my_shift_reg
```

```
set_instance_assignment -name RAMSTYLE_ATTRIBUTE LOGIC -to ram
```

You can also specify the maximum depth of memory blocks for RAM or ROM inference in RTL. Specify the `max_depth` synthesis attribute to the declaration of a variable that represents a RAM or ROM in your design file. For example:

```
// Limit the depth of the memory blocks implement "ram" to 512
// This forces the Quartus Prime software to use two M512 blocks instead of one
M4K block to implement this RAM
(* max_depth = 512 *) reg [7:0] ram[0:1023];
```

Related Links

Advanced Synthesis Settings on page 207
The following section is a quick reference of all Advanced Synthesis Settings.

4.4.1.5 Single-Clock Synchronous RAM with Old Data Read-During-Write Behavior

The code examples in this section show Verilog HDL and VHDL code that infers simple dual-port, single-clock synchronous RAM. Single-port RAM blocks use a similar coding style.

The read-during-write behavior in these examples is to read the old data at the memory address. For best performance in MLAB memories, use the appropriate attribute so that your design does not depend on the read data during a write operation. The simple dual-port RAM code samples map directly into Intel synchronous memory.

Single-port versions of memory blocks (that is, using the same read address and write address signals) allow better RAM utilization than dual-port memory blocks, depending on the device family. Refer to the appropriate device handbook for recommendations on your target device.

Example 14. Verilog HDL Single-Clock, Simple Dual-Port Synchronous RAM with Old Data Read-During-Write Behavior

```
module single_clk_ram(
    output reg [7:0] q,
    input [7:0] d,
    input [4:0] write_address, read_address,
    input we, clk
);
    reg [7:0] mem [31:0];

    always @ (posedge clk) begin
        if (we)
            mem[write_address] <= d;
        q <= mem[read_address]; // q doesn't get d in this clock cycle
    end
endmodule
```

Example 15. VHDL Single-Clock, Simple Dual-Port Synchronous RAM with Old Data Read-During-Write Behavior

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

ENTITY single_clock_ram IS
    PORT (
        clock: IN STD_LOGIC;
        data: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
        write_address: IN INTEGER RANGE 0 to 31;
        read_address: IN INTEGER RANGE 0 to 31;
        we: IN STD_LOGIC;
        q: OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
    );
END single_clock_ram;

ARCHITECTURE rtl OF single_clock_ram IS
    TYPE MEM IS ARRAY(0 TO 31) OF STD_LOGIC_VECTOR(7 DOWNTO 0);
    SIGNAL ram_block: MEM;
BEGIN
    PROCESS (clock)
    BEGIN
        IF (rising_edge(clock)) THEN
            IF (we = '1') THEN
                ram_block(write_address) <= data;
            END IF;
            q <= ram_block(read_address);
            -- VHDL semantics imply that q doesn't get data
            -- in this clock cycle
        END IF;
    END PROCESS;
END rtl;
```

4.4.1.6 Single-Clock Synchronous RAM with New Data Read-During-Write Behavior

The examples in this section describe RAM blocks in which the read-during-write behavior returns the new value being written at the memory address.

To implement this behavior in the target device, synthesis tools add bypass logic around the RAM block. This bypass logic increases the area utilization of the design, and decreases the performance if the RAM block is part of the design's critical path. If the device memory supports new data read-during-write behavior when in single-port mode (same clock, same read address, and same write address), the Verilog memory block doesn't require any bypass logic. Refer to the appropriate device handbook for specifications on your target device.

The following examples use a blocking assignment for the write so that the data is assigned intermediately.

Example 16. Verilog HDL Single-Clock, Simple Dual-Port Synchronous RAM with New Data Read-During-Write Behavior

```
module single_clock_wr_ram(
    output reg [7:0] q,
    input [7:0] d,
    input [6:0] write_address, read_address,
    input we, clk
);
    reg [7:0] mem [127:0];

    always @ (posedge clk) begin
        if (we)
            mem[write_address] = d;
        q = mem[read_address]; // q does get d in this clock
                               // cycle if we is high
    end
endmodule
```

Example 17. VHDL Single-Clock, Simple Dual-Port Synchronous RAM with New Data Read-During-Write Behavior:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY single_clock_ram IS
    PORT (
        clock: IN STD_LOGIC;
        data: IN STD_LOGIC_VECTOR (2 DOWNTO 0);
        write_address: IN INTEGER RANGE 0 to 31;
        read_address: IN INTEGER RANGE 0 to 31;
        we: IN STD_LOGIC;
        q: OUT STD_LOGIC_VECTOR (2 DOWNTO 0)
    );
END single_clock_ram;

ARCHITECTURE rtl OF single_clock_ram IS
    TYPE MEM IS ARRAY(0 TO 31) OF STD_LOGIC_VECTOR(2 DOWNTO 0);

BEGIN
    PROCESS (clock)
    VARIABLE ram_block: MEM;
    BEGIN
        IF (rising_edge(clock)) THEN
            IF (we = '1') THEN
                ram_block(write_address) := data;
            END IF;
```

```
            q <= ram_block(read_address);
            -- VHDL semantics imply that q doesn't get data
            -- in this clock cycle
        END IF;
    END PROCESS;
END rtl;
```

It is possible to create a single-clock RAM by using an `assign` statement to read the address of `mem` and create the output `q`. By itself, the RTL describes new data read-during-write behavior. However, if the RAM output feeds a register in another hierarchy, a read-during-write results in the old data. Synthesis tools may not infer a RAM block if the tool cannot determine which behavior is described, such as when the memory feeds a hard hierarchical partition boundary. Avoid this type of RTL.

Example 18. Avoid Verilog Coding Style with Vague read-during-write Behavior

```
reg [7:0] mem [127:0];
reg [6:0] read_address_reg;

always @ (posedge clk) begin
    if (we)
        mem[write_address] <= d;
    read_address_reg <= read_address;
end
assign q = mem[read_address_reg];
```

Example 19. Avoid VHDL Coding Style with Vague read-during-write Behavior

The following example uses a concurrent signal assignment to read from the RAM, and presents a similar behavior.

```
ARCHITECTURE rtl OF single_clock_rw_ram IS
    TYPE MEM IS ARRAY(0 TO 31) OF STD_LOGIC_VECTOR(2 DOWNTO 0);
    SIGNAL ram_block: MEM;
    SIGNAL read_address_reg: INTEGER RANGE 0 to 31;
BEGIN
    PROCESS (clock)
    BEGIN
        IF (rising_edge(clock)) THEN
            IF (we = '1') THEN
                ram_block(write_address) <= data;
            END IF;
            read_address_reg <= read_address;
        END IF;
    END PROCESS;
    q <= ram_block(read_address_reg);
END rtl;
```

4.4.1.7 Simple Dual-Port, Dual-Clock Synchronous RAM

With dual-clock designs, synthesis tools cannot accurately infer the read-during-write behavior because it depends on the timing of the two clocks within the target device. Therefore, the read-during-write behavior of the synthesized design is undefined and may differ from your original HDL code.

Example 20. Verilog HDL Simple Dual-Port, Dual-Clock Synchronous RAM

```
module simple_dual_port_ram_dual_clock
#(parameter DATA_WIDTH=8, parameter ADDR_WIDTH=6)
(
    input [(DATA_WIDTH-1):0] data,
    input [(ADDR_WIDTH-1):0] read_addr, write_addr,
    input we, read_clock, write_clock,
    output reg [(DATA_WIDTH-1):0] q
);

    // Declare the RAM variable
    reg [DATA_WIDTH-1:0] ram[2**ADDR_WIDTH-1:0];

    always @ (posedge write_clock)
    begin
        // Write
        if (we)
            ram[write_addr] <= data;
    end

    always @ (posedge read_clock)
    begin
        // Read
        q <= ram[read_addr];
    end

endmodule
```

Example 21. VHDL Simple Dual-Port, Dual-Clock Synchronous RAM

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY dual_clock_ram IS
    PORT (
        clock1, clock2: IN STD_LOGIC;
        data: IN STD_LOGIC_VECTOR (3 DOWNTO 0);
        write_address: IN INTEGER RANGE 0 to 31;
        read_address: IN INTEGER RANGE 0 to 31;
        we: IN STD_LOGIC;
        q: OUT STD_LOGIC_VECTOR (3 DOWNTO 0)
    );
END dual_clock_ram;
ARCHITECTURE rtl OF dual_clock_ram IS
    TYPE MEM IS ARRAY(0 TO 31) OF STD_LOGIC_VECTOR(3 DOWNTO 0);
    SIGNAL ram_block: MEM;
    SIGNAL read_address_reg : INTEGER RANGE 0 to 31;
BEGIN
    PROCESS (clock1)
    BEGIN
        IF (rising_edge(clock1)) THEN
            IF (we = '1') THEN
                ram_block(write_address) <= data;
            END IF;
        END IF;
    END PROCESS;
    PROCESS (clock2)
    BEGIN
        IF (rising_edge(clock2)) THEN
            q <= ram_block(read_address_reg);
            read_address_reg <= read_address;
        END IF;
    END PROCESS;
END rtl;
```

Related Links

Check Read-During-Write Behavior on page 110
Ensure the read-during-write behavior of the memory block described in your HDL design is consistent with your target device architecture.

4.4.1.8 True Dual-Port Synchronous RAM

The code examples in this section show Verilog HDL and VHDL code that infers true dual-port synchronous RAM. Different synthesis tools may differ in their support for these types of memories.

Intel FPGA synchronous memory blocks have two independent address ports, allowing for operations on two unique addresses simultaneously. A read operation and a write operation can share the same port if they share the same address.

The Quartus Prime software infers true dual-port RAMs in Verilog HDL and VHDL, with the following characteristics:

- Any combination of independent read or write operations in the same clock cycle.
- At most two unique port addresses.
- In one clock cycle, with one or two unique addresses, they can perform:
  - Two reads and one write
  - Two writes and one read
  - Two writes and two reads

In the synchronous RAM block architecture, there is no priority between the two ports. Therefore, if you write to the same location on both ports at the same time, the result is indeterminate in the device architecture. You must ensure your HDL code does not imply priority for writes to the memory block, if you want the design to be implemented in a dedicated hardware memory block. For example, if both ports are defined in the same process block, the code is synthesized and simulated sequentially so that there is a priority between the two ports. If your code does imply a priority, the logic cannot be implemented in the device RAM blocks and is implemented in regular logic cells. You must also consider the read-during-write behavior of the RAM block to ensure that it can be mapped directly to the device RAM architecture.

When a read and write operation occurs on the same port for the same address, the read operation may behave as follows:

- **Read new data**—Arria 10 devices support this behavior.
- **Read old data**—Not supported.

When a read and write operation occurs on different ports for the same address (also known as mixed port), the read operation may behave as follows:

- **Read new data**—Quartus Prime Pro Edition synthesis supports this mode by creating bypass logic around the synchronous memory block.
- **Read old data**—Arria 10 and Cyclone 10 devices support this behavior.
- **Read don't care**—Synchronous memory blocks support this behavior in simple dual-port mode.

The Verilog HDL single-clock code sample maps directly into synchronous Arria 10 memory blocks. When a read and write operation occurs on the same port for the same address, the new data being written to the memory is read. When a read and

write operation occurs on different ports for the same address, the old data in the memory is read. Simultaneous writes to the same location on both ports results in indeterminate behavior.

If you generate a dual-clock version of this design describing the same behavior, the inferred memory in the target device presents undefined mixed port read-during-write behavior, because it depends on the relationship between the clocks.

Example 22. Verilog HDL True Dual-Port RAM with Single Clock

```
module true_dual_port_ram_single_clock
#(parameter DATA_WIDTH = 8, ADDR_WIDTH = 6)
(
    input [(DATA_WIDTH-1):0] data_a, data_b,
    input [(ADDR_WIDTH-1):0] addr_a, addr_b,
    input we_a, we_b, clk,
    output reg [(DATA_WIDTH-1):0] q_a, q_b
);


    // Declare the RAM variable
    reg [DATA_WIDTH-1:0] ram[2**ADDR_WIDTH-1:0];

    always @ (posedge clk)
    begin // Port a
        if (we_a)
        begin
            ram[addr_a] <= data_a;
            q_a <= data_a;
        end
        else
            q_a <= ram[addr_a];
    end
    always @ (posedge clk)
    begin // Port b
        if (we_b)
        begin
            ram[addr_b] <= data_b;
            q_b <= data_b;
        end
        else
            q_b <= ram[addr_b];
    end
endmodule
```

Example 23. VHDL Read Statement Example

```
-- Port A
process(clk)
    begin
    if(rising_edge(clk)) then
        if(we_a = '1') then
            ram(addr_a) := data_a;
        end if;
        q_a <= ram(addr_a);
    end if;
end process;

-- Port B
process(clk)
    begin
    if(rising_edge(clk)) then
        if(we_b = '1') then
            ram(addr_b) := data_b;
        end if;
```

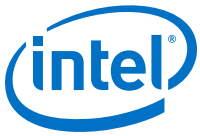
```
        q_b <= ram(addr_b);
    end if;
end process;
```

The VHDL single-clock code sample maps directly into Intel FPGA synchronous memory. When a read and write operation occurs on the same port for the same address, the new data writing to the memory is read. When a read and write operation occurs on different ports for the same address, the behavior results in old data for Arria 10 and Cyclone 10 devices is undefined. Simultaneous write operations to the same location on both ports results in indeterminate behavior.

If you generate a dual-clock version of this design describing the same behavior, the memory in the target device presents undefined mixed port read-during-write behavior because it depends on the relationship between the clocks.

Example 24. VHDL True Dual-Port RAM with Single Clock

```
LIBRARY ieee;
use ieee.std_logic_1164.all;

entity true_dual_port_ram_single_clock is
    generic (
        DATA_WIDTH : natural := 8;
        ADDR_WIDTH : natural := 6
    );

port (
    clk : in std_logic;
    addr_a : in natural range 0 to 2**ADDR_WIDTH - 1;
    addr_b : in natural range 0 to 2**ADDR_WIDTH - 1;
    data_a : in std_logic_vector((DATA_WIDTH-1) downto 0);
    data_b : in std_logic_vector((DATA_WIDTH-1) downto 0);
    we_a : in std_logic := '1';
    we_b : in std_logic := '1';
    q_a : out std_logic_vector((DATA_WIDTH -1) downto 0);
    q_b : out std_logic_vector((DATA_WIDTH -1) downto 0)
);
end true_dual_port_ram_single_clock;

architecture rtl of true_dual_port_ram_single_clock is
    -- Build a 2-D array type for the RAM
    subtype word_t is std_logic_vector((DATA_WIDTH-1) downto 0);

    type memory_t is array((2**ADDR_WIDTH - 1) downto 0) of word_t;
    -- Declare the RAM signal.
    signal ram : memory_t;

begin
    process(clk)
    begin
        if(rising_edge(clk)) then -- Port A
            if(we_a = '1') then
                ram(addr_a) <= data_a;
                -- Read-during-write on same port returns NEW data
                q_a <= data_a;
            else
              -- Read-during-write on mixed port returns OLD data
                q_a <= ram(addr_a);
            end if;
        end if;
    end process;

    process(clk)
    begin
        if(rising_edge(clk)) then -- Port B
            if(we_b = '1') then
```



```
                ram(addr_b) <= data_b;
                -- Read-during-write on same port returns NEW data
                q_b <= data_b;
            else
              -- Read-during-write on mixed port returns OLD data
                q_b <= ram(addr_b);
            end if;
        end if;
    end process;
end rtl;
```

The port behavior inferred in the Quartus Prime software for the above example is:

```
PORT_A_READ_DURING_WRITE_MODE = "new_data_no_nbe_read"
PORT_B_READ_DURING_WRITE_MODE = "new_data_no_nbe_read"
MIXED_PORT_FEED_THROUGH_MODE = "old"
```

Related Links

Guideline: Customize Read-During-Write Behavior
In *Intel Arria 10 Core Fabric and General Purpose I/Os Handbook*

4.4.1.9 Mixed-Width Dual-Port RAM

The RAM code examples in this section show SystemVerilog and VHDL code that infers RAM with data ports with different widths.

Verilog-1995 doesn't support mixed-width RAMs because the standard lacks a multi-dimensional array to model the different read width, write width, or both. Verilog-2001 doesn't support mixed-width RAMs because this type of logic requires multiple packed dimensions. Different synthesis tools may differ in their support for these memories. This section describes the inference rules for Quartus Prime Pro Edition synthesis.

The first dimension of the multi-dimensional packed array represents the ratio of the wider port to the narrower port. The second dimension represents the narrower port width. The read and write port widths must specify a read or write ratio supported by the memory blocks in the target device. Otherwise, the synthesis tool does not infer a RAM.

Refer to the Quartus Prime HDL templates for parameterized examples with supported combinations of read and write widths. You can also find examples of true dual port RAMs with two mixed-width read ports and two mixed-width write ports.

Example 25. SystemVerilog Mixed-Width RAM with Read Width Smaller than Write Width

```
module mixed_width_ram    // 256x32 write and 1024x8 read
(
        input [7:0] waddr,
        input [31:0] wdata,
        input we, clk,
        input [9:0] raddr,
        output logic [7:0] q
);
    logic [3:0][7:0] ram[0:255];
    always_ff@(posedge clk)
        begin
            if(we) ram[waddr] <= wdata;
            q <= ram[raddr / 4][raddr % 4];
        end
endmodule : mixed_width_ram
```

Example 26. SystemVerilog Mixed-Width RAM with Read Width Larger than Write Width

```
module mixed_width_ram     // 1024x8 write and 256x32 read
(
        input [9:0] waddr,
        input [31:0] wdata,
        input we, clk,
        input [7:0] raddr,
        output logic [9:0] q
);
    logic [3:0][7:0] ram[0:255];
    always_ff@(posedge clk)
         begin
            if(we) ram[waddr / 4][waddr % 4] <= wdata;
            q <= ram[raddr];
         end
endmodule : mixed_width_ram
```

Example 27. VHDL Mixed-Width RAM with Read Width Smaller than Write Width

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

package ram_types is
    type word_t is array (0 to 3) of std_logic_vector(7 downto 0);
    type ram_t is array (0 to 255) of word_t;
end ram_types;

library ieee;
use ieee.std_logic_1164.all;
library work;
use work.ram_types.all;

entity mixed_width_ram is
    port (
        we, clk : in  std_logic;
        waddr   : in  integer range 0 to 255;
        wdata   : in  word_t;
        raddr   : in  integer range 0 to 1023;
        q       : out std_logic_vector(7 downto 0));
end mixed_width_ram;

architecture rtl of mixed_width_ram is
    signal ram : ram_t;
begin  -- rtl
    process(clk, we)
    begin
        if(rising_edge(clk)) then
            if(we = '1') then
                ram(waddr) <= wdata;
            end if;
            q <= ram(raddr / 4 )(raddr mod 4);
        end if;
    end process;
end rtl;
```

Example 28. VHDL Mixed-Width RAM with Read Width Larger than Write Width

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

package ram_types is
    type word_t is array (0 to 3) of std_logic_vector(7 downto 0);
    type ram_t is array (0 to 255) of word_t;
end ram_types;
```

```
library ieee;
use ieee.std_logic_1164.all;
library work;
use work.ram_types.all;

entity mixed_width_ram is
    port (
        we, clk : in  std_logic;
        waddr   : in  integer range 0 to 1023;
        wdata   : in  std_logic_vector(7 downto 0);
        raddr   : in  integer range 0 to 255;
        q       : out word_t);
end mixed_width_ram;

architecture rtl of mixed_width_ram is
    signal ram : ram_t;
begin  -- rtl
    process(clk, we)
    begin
        if(rising_edge(clk)) then
            if(we = '1') then
                ram(waddr / 4)(waddr mod 4) <= wdata;
            end if;
            q <= ram(raddr);
        end if;
    end process;
end rtl;
```

4.4.1.10 RAM with Byte-Enable Signals

The RAM code examples in this section show SystemVerilog and VHDL code that infers RAM with controls for writing single bytes into the memory word, or byte-enable signals.

Synthesis models byte-enable signals by creating write expressions with two indexes, and writing part of a RAM "word." With these implementations, you can also write more than one byte at once by enabling the appropriate byte enables.

Verilog-1995 doesn't support mixed-width RAMs because the standard lacks a multi-dimensional array to model the different read width, write width, or both. Verilog-2001 doesn't support mixed-width RAMs because this type of logic requires multiple packed dimensions. Different synthesis tools may differ in their support for these memories. This section describes the inference rules for Quartus Prime Pro Edition synthesis.

Refer to the Quartus Prime HDL templates for parameterized examples that you can use for different address widths, and true dual port RAM examples with two read ports and two write ports.

Example 29. SystemVerilog Simple Dual-Port Synchronous RAM with Byte Enable

```
module byte_enabled_simple_dual_port_ram
(
    input we, clk,
    input [5:0] waddr, raddr, // address width = 6
    input [3:0] be,         // 4 bytes per word
    input [31:0] wdata,      // byte width = 8, 4 bytes per word
    output reg [31:0] q      // byte width = 8, 4 bytes per word
);
    // use a multi-dimensional packed array
    //to model individual bytes within the word
   logic [3:0][7:0] ram[0:63];    // # words = 1 << address width

   always_ff@(posedge clk)
   begin
```

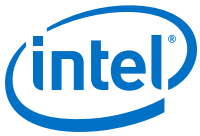
```
        if(we) begin
           if(be[0]) ram[waddr][0] <= wdata[7:0];
           if(be[1]) ram[waddr][1] <= wdata[15:8];
           if(be[2]) ram[waddr][2] <= wdata[23:16];
        if(be[3]) ram[waddr][3] <= wdata[31:24];
        end
        q <= ram[raddr];
   end
endmodule
```

Example 30. VHDL Simple Dual-Port Synchronous RAM with Byte Enable

```
library ieee;
use ieee.std_logic_1164.all;
library work;

entity byte_enabled_simple_dual_port_ram is
port (
    we, clk : in  std_logic;
    waddr, raddr : in  integer range 0 to 63 ;     -- address width = 6
    be      : in  std_logic_vector (3 downto 0);   -- 4 bytes per word
    wdata   : in  std_logic_vector(31 downto 0);   -- byte width = 8
    q       : out std_logic_vector(31 downto 0) ); -- byte width = 8
end byte_enabled_simple_dual_port_ram;

architecture rtl of byte_enabled_simple_dual_port_ram is
  --  build up 2D array to hold the memory
    type word_t is array (0 to 3) of std_logic_vector(7 downto 0);
    type ram_t is array (0 to 63) of word_t;

    signal ram : ram_t;
    signal q_local : word_t;

    begin  -- Re-organize the read data from the RAM to match the output
        unpack: for i in 0 to 3 generate
            q(8*(i+1) - 1 downto 8*i) <= q_local(i);
    end generate unpack;

    process(clk)
    begin
        if(rising_edge(clk)) then
            if(we = '1') then
                if(be(0) = '1') then
                    ram(waddr)(0) <= wdata(7 downto 0);
                end if;
                if be(1) = '1' then
                    ram(waddr)(1) <= wdata(15 downto 8);
                end if;
                if be(2) = '1' then
                    ram(waddr)(2) <= wdata(23 downto 16);
                end if;
                if be(3) = '1' then
                    ram(waddr)(3) <= wdata(31 downto 24);
                end if;
            end if;
            q_local <= ram(raddr);
        end if;
    end process;
end rtl;
```

4.4.1.11 Specifying Initial Memory Contents at Power-Up

Your synthesis tool may offer various ways to specify the initial contents of an inferred memory. There are slight power-up and initialization differences between dedicated RAM blocks and the MLAB memory, due to the continuous read of the MLAB.



Intel FPGA dedicated RAM block outputs always power-up to zero, and are set to the initial value on the first read. For example, if address `0` is pre-initialized to FF, the RAM block powers up with the output at `0`. A subsequent read after power-up from address `0` outputs the pre-initialized value of FF. Therefore, if a RAM powers up and an enable (read enable or clock enable) is held low, the power-up output of `0` maintains until the first valid read cycle. The synthesis tool implements MLAB using registers that power-up to `0`, but initialize to their initial value immediately at power-up or reset. Therefore, the initial value is seen, regardless of the enable status. The Quartus Prime software maps inferred memory to MLABs when the HDL code specifies an appropriate `ramstyle` attribute.

In Verilog HDL, you can use an initial block to initialize the contents of an inferred memory. Quartus Prime Pro Edition synthesis automatically converts the initial block into a Memory Initialization File (`.mif`) for the inferred RAM.

Example 31. Verilog HDL RAM with Initialized Contents

```
module ram_with_init(
   output reg [7:0] q,
   input [7:0] d,
   input [4:0] write_address, read_address,
   input we, clk
);
   reg [7:0] mem [0:31];
   integer i;

   initial begin
      for (i = 0; i < 32; i = i + 1)
         mem[i] = i[7:0];
   end

   always @ (posedge clk) begin
      if (we)
         mem[write_address] <= d;
      q <= mem[read_address];
   end
endmodule
```

Quartus Prime Pro Edition synthesis and other synthesis tools also support the `$readmemb` and `$readmemh` attributes. These attributes allow RAM initialization and ROM initialization work identically in synthesis and simulation.

Example 32. Verilog HDL RAM Initialized with the readmemb Command

```
reg [7:0] ram[0:15];
initial
begin
    $readmemb("ram.txt", ram);
end
```

In VHDL, you can initialize the contents of an inferred memory by specifying a default value for the corresponding signal. Quartus Prime Pro Edition synthesis automatically converts the default value into a `.mif` file for the inferred RAM.

Example 33. VHDL RAM with Initialized Contents

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
use ieee.numeric_std.all;
```

```
ENTITY ram_with_init IS
    PORT(
            clock: IN STD_LOGIC;
            data: IN UNSIGNED (7 DOWNTO 0);
            write_address: IN integer RANGE 0 to 31;
            read_address: IN integer RANGE 0 to 31;
            we: IN std_logic;
            q: OUT UNSIGNED (7 DOWNTO 0));
END;

ARCHITECTURE rtl OF ram_with_init IS

    TYPE MEM IS ARRAY(31 DOWNTO 0) OF unsigned(7 DOWNTO 0);
    FUNCTION initialize_ram
        return MEM is
        variable result : MEM;
    BEGIN
        FOR i IN 31 DOWNTO 0 LOOP
            result(i) := to_unsigned(natural(i), natural'(8));
        END LOOP;
        RETURN result;
    END initialize_ram;

    SIGNAL ram_block : MEM := initialize_ram;
BEGIN
    PROCESS (clock)
    BEGIN
        IF (rising_edge(clock)) THEN
            IF (we = '1') THEN
            ram_block(write_address) <= data;
            END IF;
            q <= ram_block(read_address);
        END IF;
    END PROCESS;
END rtl;
```

4.4.2 Inferring ROM Functions from HDL Code

Synthesis tools infer ROMs when a CASE statement exists in which a value is set to a constant for every choice in the CASE statement.

Because small ROMs typically achieve the best performance when they are implemented using the registers in regular logic, each ROM function must meet a minimum size requirement for inference and placement in memory.

For device architectures with synchronous RAM blocks, to infer a ROM block, synthesis must use registers for either the address or the output. When your design uses output registers, synthesis implements registers from the input registers of the RAM block without affecting the functionality of the ROM. If you register the address, the power-up state of the inferred ROM can be different from the HDL design. In this scenario, Quartus Prime synthesis issues a warning.

The following ROM examples map directly to the Intel FPGA memory architecture.

Example 34. Verilog HDL Synchronous ROM

```
module sync_rom (clock, address, data_out);
    input clock;
    input [7:0] address;
    output reg [5:0] data_out;
    reg [5:0] data_out;

    always @ (posedge clock)
```

```
    begin
        case (address)
            8'b00000000: data_out = 6'b101111;
            8'b00000001: data_out = 6'b110110;
            ...
            8'b11111110: data_out = 6'b000001;
            8'b11111111: data_out = 6'b101010;
        endcase
    end
endmodule
```

Example 35. VHDL Synchronous ROM

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

ENTITY sync_rom IS
    PORT (
        clock: IN STD_LOGIC;
        address: IN STD_LOGIC_VECTOR(7 downto 0);
        data_out: OUT STD_LOGIC_VECTOR(5 downto 0)
    );
END sync_rom;

ARCHITECTURE rtl OF sync_rom IS
BEGIN
PROCESS (clock)
    BEGIN
    IF rising_edge (clock) THEN
        CASE address IS
            WHEN "00000000" => data_out <= "101111";
            WHEN "00000001" => data_out <= "110110";
            ...
            WHEN "11111110" => data_out <= "000001";
            WHEN "11111111" => data_out <= "101010";
            WHEN OTHERS     => data_out <= "101111";
        END CASE;
    END IF;
    END PROCESS;
END rtl;
```

Example 36. Verilog HDL Dual-Port Synchronous ROM Using readmemb

```
module dual_port_rom
#(parameter data_width=8, parameter addr_width=8)
(
    input [(addr_width-1):0] addr_a, addr_b,
    input clk,
    output reg [(data_width-1):0] q_a, q_b
);
    reg [data_width-1:0] rom[2**addr_width-1:0];

    initial // Read the memory contents in the file
             //dual_port_rom_init.txt.
    begin
        $readmemb("dual_port_rom_init.txt", rom);
    end

    always @ (posedge clk)
    begin
        q_a <= rom[addr_a];
        q_b <= rom[addr_b];
    end
endmodule
```

Example 37. VHDL Dual-Port Synchronous ROM Using Initialization Function

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity dual_port_rom is
    generic (
        DATA_WIDTH : natural := 8;
        ADDR_WIDTH : natural := 8
    );
    port (
        clk     : in std_logic;
        addr_a  : in natural range 0 to 2**ADDR_WIDTH - 1;
        addr_b  : in natural range 0 to 2**ADDR_WIDTH - 1;
        q_a     : out std_logic_vector((DATA_WIDTH -1) downto 0);
        q_b     : out std_logic_vector((DATA_WIDTH -1) downto 0)
    );
end entity;

architecture rtl of dual_port_rom is
    -- Build a 2-D array type for the ROM
    subtype word_t is std_logic_vector((DATA_WIDTH-1) downto 0);
    type memory_t is array(2**ADDR_WIDTH - 1 downto 0) of word_t;

    function init_rom
        return memory_t is
        variable tmp : memory_t := (others => (others => '0'));
    begin
        for addr_pos in 0 to 2**ADDR_WIDTH - 1 loop
            -- Initialize each address with the address itself
            tmp(addr_pos) := std_logic_vector(to_unsigned(addr_pos, DATA_WIDTH));
        end loop;
        return tmp;
    end init_rom;

    -- Declare the ROM signal and specify a default initialization value.
    signal rom : memory_t := init_rom;
begin
    process(clk)
    begin
    if (rising_edge(clk)) then
        q_a <= rom(addr_a);
        q_b <= rom(addr_b);
    end if;
    end process;
end rtl;
```

4.4.3 Inferring Shift Registers in HDL Code

To infer shift registers in Arria 10 devices, synthesis tools detect a group of shift registers of the same length, and convert them to an Intel FPGA shift register IP core.

For detection, all shift registers must have the following characteristics:

- Use the same clock and clock enable
- No other secondary signals
- Equally spaced taps that are at least three registers apart

Synthesis recognizes shift registers only for device families with dedicated RAM blocks. Quartus Prime Pro Edition synthesis uses the following guidelines:

- The Quartus Prime software determines whether to infer the Intel FPGA shift register IP core based on the width of the registered bus (*W*), the length between each tap (*L*), or the number of taps (*N*).
- If the **Auto Shift Register Recognition** option is set to **Auto**, Quartus Prime Pro Edition synthesis determines which shift registers are implemented in RAM blocks for logic by using:
  - The **Optimization Technique** setting
  - Logic and RAM utilization information about the design
  - Timing information from **Timing-Driven Synthesis**
- If the registered bus width is one ($W = 1$), Quartus Prime synthesis infers shift register IP if the number of taps times the length between each tap is greater than or equal to 64 ($N \times L > 64$).
- If the registered bus width is greater than one ($W > 1$), and the registered bus width times the number of taps times the length between each tap is greater than or equal to 32 ($W \times N \times L > 32$), the Quartus Prime synthesis infers Intel FPGA shift register IP core.
- If the length between each tap (*L*) is not a power of two, Quartus Prime synthesis needs external logic (LEs or ALMs) to decode the read and write counters, because of different sizes of shift registers. This extra decode logic eliminates the performance and utilization advantages of implementing shift registers in memory.

The registers that Quartus Prime synthesis maps to the Intel FPGA shift register IP core, and places in RAM are not available in a Verilog HDL or VHDL output file for simulation tools, because their node names do not exist after synthesis.

Note: The Compiler cannot implement a shift register that uses a `shift enable` signal into MLAB memory; instead, the Compiler uses dedicated RAM blocks. To control the type of memory structure that implements the shift register, use the `ramstyle` attribute.

4.4.3.1 Simple Shift Register

The examples in this section show a simple, single-bit wide, 67-bit long shift register.

Quartus Prime synthesis implements the register (`W = 1` and `M = 67`) in an ALTSHIFT_TAPS IP core for supported devices and maps it to RAM in supported devices, which may be placed in dedicated RAM blocks or MLAB memory. If the length of the register is less than 67 bits, Quartus Prime synthesis implements the shift register in logic.

Example 38. Verilog HDL Single-Bit Wide, 64-Bit Long Shift Register

```
module shift_1x67 (clk, shift, sr_in, sr_out);
    input clk, shift;
    input sr_in;
    output sr_out;

    reg [66:0] sr;

    always @ (posedge clk)
    begin
        if (shift == 1'b1)
        begin
```

```
            sr[66:1] <= sr[65:0];
            sr[0] <= sr_in;
        end
    end
    assign sr_out = sr[65];
endmodule
```

Example 39. VHDL Single-Bit Wide, 64-Bit Long Shift Register

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.all;
ENTITY shift_1x67 IS
    PORT (
        clk: IN STD_LOGIC;
        shift: IN STD_LOGIC;
        sr_in: IN STD_LOGIC;
        sr_out: OUT STD_LOGIC
    );
END shift_1x67;

ARCHITECTURE arch OF shift_1x67 IS
    TYPE sr_length IS ARRAY (66 DOWNTO 0) OF STD_LOGIC;
    SIGNAL sr: sr_length;
BEGIN
    PROCESS (clk)
        BEGIN
        IF (rising_edge(clk)) THEN
            IF (shift = '1') THEN
            sr(66 DOWNTO 1) <= sr(65 DOWNTO 0);
            sr(0) <= sr_in;
            END IF;
        END IF;
    END PROCESS;
    sr_out <= sr(65);
END arch;
```

4.4.3.2 Shift Register with Evenly Spaced Taps

The following examples show a Verilog HDL and VHDL 8-bit wide, 64-bit long shift register (`W > 1` and `M = 64`) with evenly spaced taps at 15, 31, and 47.

The synthesis software implements this function in a single ALTSHIFT_TAPS IP core and maps it to RAM in supported devices, which is allowed placement in dedicated RAM blocks or MLAB memory.

Example 40. Verilog HDL 8-Bit Wide, 64-Bit Long Shift Register with Evenly Spaced Taps

```
module top (clk, shift, sr_in, sr_out, sr_tap_one, sr_tap_two,
                sr_tap_three );
    input clk, shift;
    input [7:0] sr_in;
    output [7:0] sr_tap_one, sr_tap_two, sr_tap_three, sr_out;
    reg [7:0] sr [64:0];
    integer n;
    always @ (posedge clk)
        begin
        if (shift == 1'b1)
            begin
            for (n = 64; n>0; n = n-1)
                begin
                sr[n] <= sr[n-1];
                end
            sr[0] <= sr_in;
        end
    end
```

```
    assign sr_tap_one = sr[16];
    assign sr_tap_two = sr[32];
    assign sr_tap_three = sr[48];
    assign sr_out = sr[64];
endmodule
```

4.5 Register and Latch Coding Guidelines

This section provides device-specific coding recommendations for Intel registers and latches. Understanding the architecture of the target Intel device helps ensure that your RTL produces the expected results and achieves the optimal quality of results.

4.5.1 Register Power-Up Values

Registers in the device core always power-up to a low (0) logic level on all Intel FPGA devices. However, If your design specifies a power-up level other than 0, synthesis tools can implement logic that causes registers to behave as if they were powering up to a high (1) logic level.

If your design uses a `preset` signal, but your device does not support presets in the register architecture, synthesis may convert the `preset` signal to a `clear` signal, which requires to perform a NOT gate push-back optimization. NOT gate push-back adds an inverter to the input and the output of the register, so that the reset and power-up conditions appear high, and the device operates as expected. In this case, your synthesis tool may issue a message about the power-up condition. The register itself powers up low, but since the register output inverts, the signal that arrives at all destinations is high.

Due to these effects, if you specify a non-zero reset value, your synthesis tool may use the asynchronous clear (`aclr`) signals available on the registers to implement the high bits with NOT gate push-back. In that case, the registers look as though they power-up to the specified reset value.

When an asynchronous load (`aload`) signal is available in the device registers, your synthesis tools can implement a reset of `1` or `0` value by using an asynchronous load of `1` or `0`. When the synthesis tool uses a `load` signal, it is not performing NOT gate push-back, so the registers power-up to a `0` logic level.

For additional details, refer to the appropriate device family handbook.

Optionally use an explicit reset signal for the design, which forces all registers into their appropriate values after reset. Use this practice to reset the device after power-up to restore the proper state.

Make your design more stable and avoid potential glitches by synchronizing external or combinational logic of the device architecture before you drive the asynchronous control ports of registers.

Related Links

Recommended Design Practices on page 155
This chapter provides design recommendations for Intel FPGA devices.

4.5.1.1 Specifying a Power-Up Value

To specify a particular power-up condition for your design, use the synthesis options available in your synthesis tool. Quartus Prime Pro Edition synthesis provides the **Power-Up Level** logic option.

You can also specify the power-up level with an `altera_attribute` assignment in your source code. This attribute forces synthesis to perform NOT gate push-back, because synthesis tools cannot actually change the power-up states of core registers.

You can apply the **Power-Up Level** logic option to a specific register, or to a design entity, module, or subdesign. When you assign this option, every register in that block receives the value. Registers power up to `0` by default. Therefore, you can use this assignment to force all registers to power-up to `1` using NOT gate push-back.

Setting the **Power-Up Level** to a logic level of **high** for a large design entity could degrade the quality of results due to the number of inverters that requires. In some situations, this design style causes issues due to `enable` signal inference or secondary control logic inference. It may also be more difficult to migrate this type of designs.

Some synthesis tools can also read the default or initial values for registered signals and implement this behavior in the device. For example, Quartus Prime Pro Edition synthesis converts default values for registered signals into **Power-Up Level** settings. When the Quartus Prime software reads the default values, the synthesized behavior matches the power-up state of the HDL code during a functional simulation.

Example 41. Verilog Register with High Power-Up Value

```
reg q = 1'b1; //q has a default value of '1'

always @ (posedge clk)
begin
    q <= d;
end
```

Example 42. VHDL Register with High Power-Up Level

```
SIGNAL q : STD_LOGIC := '1'; -- q has a default value of '1'

PROCESS (clk, reset)
BEGIN
    IF (rising_edge(clk)) THEN
        q <= d;
    END IF;
END PROCESS;
```

Your design may contain undeclared default power-up conditions based on signal type. If you declare a VHDL register signal as an integer, Quartus Prime synthesis uses the left end of the integer range as the power-up value. For the default signed integer type, the default power-up value is the highest magnitude negative integer (`100…001`). For an unsigned integer type, the default power-up value is `0`.

Note: If the target device architecture does not support two asynchronous control signals, such as `aclr` and `aload`, you cannot set a different power-up state and reset state. If the NOT gate push-back algorithm creates logic to set a register to `1`, that register powers-up high. If you set a different power-up condition through a synthesis attribute or initial value, synthesis ignores the power-up level.

4.5.2 Secondary Register Control Signals Such as Clear and Clock Enable

The registers in Intel FPGAs provide a number of secondary control signals. Use these signals to implement control logic for each register without using extra logic cells. Intel FPGA device families vary in their support for secondary signals, so consult the device family data sheet to verify which signals are available in your target device.

To make the most efficient use of the signals in the device, ensure that HDL code matches the device architecture as closely as possible. The control signals have a certain priority due to the nature of the architecture. Your HDL code must follow that priority where possible.

Your synthesis tool can emulate any control signals using regular logic, so achieving functionally correct results is always possible. However, if your design requirements allow flexibility in controlling use and priority of control signals, match your design to the target device architecture to achieve the most efficient results. If the priority of the signals in your design is not the same as that of the target architecture, you may require extra logic to implement the control signals. This extra logic uses additional device resources, and can cause additional delays for the control signals.

In certain cases, using logic other than the dedicated control logic in the device architecture can have a larger impact. For example, the `clock enable` signal has priority over the synchronous `reset` or `clear` signal in the device architecture. The `clock enable` turns off the clock line in the LAB, and the `clear` signal is synchronous. Therefore, in the device architecture, the synchronous clear takes effect only when a clock edge occurs.

If you define a register with a synchronous `clear` signal that has priority over the `clock enable` signal, Quartus Prime synthesis emulates the clock enable functionality using data inputs to the registers. You cannot apply a Clock Enable Multicycle constraint, because the emulated functionality does not use the `clock enable` port of the register. In this case, using a different priority causes unexpected results with an assignment to the `clock enable` signal.

The signal order is the same for all Intel FPGA device families. However, not all device families provide every signal. The priority order is:

1. Asynchronous Clear (`clrn`)—highest priority
2. Enable (`ena`)
3. Synchronous Clear (`sclr`)
4. Synchronous Load (`sload`)
5. Data In (`data`)—lowest priority

The priority order for secondary control signals in Intel FPGA devices differs from the order for other vendors' FPGA devices. If your design requirements are flexible regarding priority, verify that the secondary control signals meet design performance requirements when migrating designs between FPGA vendors. To achieve the best results. try to match your target device architecture.

Example 43. Verilog D-type Flipflop bus with Secondary Signals

This module uses all Arria 10 DFF secondary signals: `clrn`, `ena`, `sclr`, and `sload`. Note that it instantiates 8-bit bus of DFFs rather than a single DFF, because synthesis infers some secondary signals only if there are multiple DFFs with the same secondary signal.

```
module top(clk, clrn, sclr, sload, ena, data, sdata, q);
    input clk, clrn, sclr, sload, ena;
    input [7:0] data, sdata;
    output [7:0] q;
    reg [7:0] q;
    always @ (posedge clk or posedge clrn)
        begin
        if (clrn)
            q <= 8'b0;
        else if (ena)
            begin
            if (sclr)
                q <= 8'b0;
            else if (!sload)
                q <= data;
            else
                q <= sdata;
            end
        end
endmodule
```

Related Links

Clock Enable Multicycle
In *Quartus Prime TimeQuest Timing Analyzer Cookbook*

4.5.3 Latches

A latch is a small combinational loop that holds the value of a signal until a new value is assigned.
Synthesis tools can infer latches from HDL code when you did not intend to use a latch. If you do intend to infer a latch, it is important to infer it correctly to guarantee correct device operation.

Note: Design without the use of latches whenever possible.

Related Links

Avoid Unintended Latch Inference on page 158
Avoid using latches to ensure that you can completely analyze the timing performance and reliability of your design.

4.5.3.1 Avoid Unintentional Latch Generation

When you design combinational logic, certain coding styles can create an unintentional latch. For example, when CASE or IF statements do not cover all possible input conditions, synthesis tools can infer latches to hold the output if a new output value is not assigned. Check your synthesis tool messages for references to inferred latches.

If your code unintentionally creates a latch, modify your RTL to remove the latch:

- Synthesis infers a latch when HDL code assigns a value to a signal outside of a clock edge (for example, with an asynchronous reset), but the code don't assign a value in an edge-triggered design block.
- Unintentional latches also occur when HDL code assigns a value to a signal in an edge-triggered design block, but synthesis optimizations remove that logic. For example, when a CASE or IF statement tests a condition that only evaluates to FALSE, synthesis removes any logic or signal assignment in that statement during optimization. This optimization may result in the inference of a latch for the signal.
- Omitting the final ELSE or WHEN OTHERS clause in an IF or CASE statement can also generate a latch. Don’t care (X) assignments on the default conditions are useful in preventing latch generation. For the best logic optimization, assign the default CASE or final ELSE value to don’t care (X) instead of a logic value.

In Verilog HDL designs, use the full_case attribute to treat unspecified cases as don’t care values (X). However, since the full_case attribute is synthesis-only, it can cause simulation mismatches, because simulation tools still treat the unspecified cases as latches.

Example 44. VHDL Code Preventing Unintentional Latch Creation

Without the final ELSE clause, the following code creates unintentional latches to cover the remaining combinations of the SEL inputs. When you are targeting a Stratix series device with this code, omitting the final ELSE condition can cause synthesis tools to use up to six LEs, instead of the three it uses with the ELSE statement. Additionally, assigning the final ELSE clause to 1 instead of X can result in slightly more LEs, because synthesis tools cannot perform as much optimization when you specify a constant value as opposed to a don’t care value.

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

ENTITY nolatch IS
    PORT (a,b,c: IN STD_LOGIC;
        sel: IN STD_LOGIC_VECTOR (4 DOWNTO 0);
        oput: OUT STD_LOGIC);
END nolatch;

ARCHITECTURE rtl OF nolatch IS
BEGIN
    PROCESS (a,b,c,sel) BEGIN
        IF sel = "00000" THEN
            oput <= a;
        ELSIF sel = "00001" THEN
            oput <= b;
        ELSIF sel = "00010" THEN
            oput <= c;
        ELSE                    --- Prevents latch inference
            oput <= 'X'; --/
```

```
        END IF;
    END PROCESS;
END rtl;
```

4.5.3.2 Inferring Latches Correctly

Synthesis tools can infer a latch that does not exhibit the glitch and timing hazard problems typically associated with combinational loops. Quartus Prime Pro Edition software reports latches that synthesis inferred in the **User-Specified and Inferred Latches** section of the Compilation Report. This report indicates whether or not the latch presents a timing hazard, and the total number of user-specified and inferred latches.

Note: Timing analysis does not completely model latch timing in some cases. Do not use latches unless required by your design, and you fully understand the impact of using the latches.

If a latch or combinational loop in your design doesn't appear in the **User Specified and Inferred Latches** section, it means that Quartus Prime synthesis didn't infer the latch as a safe latch, so it is not considered glitch-free.

All combinational loops listed in the **Analysis & Synthesis Logic Cells Representing Combinational Loops** table in the Compilation Report are at risk of timing hazards. These entries indicate possible problems with your design that you should investigate. However, it is possible to have a correct design that includes combinational loops. For example, it is possible that the combinational loop cannot be sensitized. This occurs when there is an electrical path in the hardware, but either:

- The designer knows that the circuit never encounters data that causes that path to be activated, or
- The surrounding logic is set up in a mutually exclusive manner that prevents that path from ever being sensitized, independent of the data input.

For 6-input LUT-based devices, Quartus Prime synthesis implements all latch inputs with a single adaptive look-up table (ALUT) in the combinational loop. Therefore, all latches in the **User-Specified and Inferred Latches** table are free of timing hazards when a single input changes.

If Quartus Prime synthesis report lists a latch as a safe latch, other optimizations, such as physical synthesis netlist optimizations in the Fitter, maintain the hazard-free performance. To ensure hazard-free behavior, only one control input can change at a time. Changing two inputs simultaneously, such as deasserting `set` and `reset` at the same time, or changing data and enable at the same time, can produce incorrect behavior in any latch.

Quartus Prime synthesis infers latches from `always` blocks in Verilog HDL and `process` statements in VHDL. However, Quartus Prime synthesis does not infer latches from continuous assignments in Verilog HDL, or concurrent signal assignments in VHDL. These rules are the same as for register inference. The Quartus Prime synthesis infers registers or flipflops only from `always` blocks and `process` statements.

Example 45. Verilog HDL Set-Reset Latch

```
module simple_latch (
   input SetTerm,
   input ResetTerm,
   output reg LatchOut
   );
   always @ (SetTerm or ResetTerm) begin
      if (SetTerm)
         LatchOut = 1'b1;
      else if (ResetTerm)
         LatchOut = 1'b0;
   end
endmodule
```

Example 46. VHDL Data Type Latch

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
ENTITY simple_latch IS
    PORT (
        enable, data    : IN STD_LOGIC;
        q               : OUT STD_LOGIC
    );
END simple_latch;
ARCHITECTURE rtl OF simple_latch IS
BEGIN
    latch : PROCESS (enable, data)
        BEGIN
        IF (enable = '1') THEN
            q <= data;
        END IF;
    END PROCESS latch;
END rtl;
```

The following example shows a Verilog HDL continuous assignment that does not infer a latch in the Quartus Prime software:

Example 47. VHDL Continuous Assignment Does Not Infer Latch

```
assign latch_out = (~en & latch_out) | (en & data);
```

The behavior of the assignment is similar to a latch, but it may not function correctly as a latch, and its timing is not analyzed as a latch. Quartus Prime Pro Edition synthesis also creates safe latches when possible for instantiations of an Altera latch IP core. Use an Altera latch IP core to define a latch with any combination of data, enable, set, and `reset` inputs. The same limitations apply for creating safe latches as for inferring latches from HDL code.

Inferring the Altera latch IP core in another synthesis tool ensures that Quartus Prime synthesis also recognizes the implementation as a latch. If a third-party synthesis tool implements a latch using the Altera latch IP core, Quartus Prime Pro Edition synthesis reports the latch in the **User-Specified and Inferred Latches** table, in the same manner as it lists latches you define in HDL source code. The coding style necessary to produce an Altera latch IP core implementation may depend on your synthesis tool. Some third-party synthesis tools list the number of Altera latch IP cores that are inferred.

The Fitter uses global routing for control signals, including signals that synthesis identifies as latch enables. In some cases the global insertion delay may decrease the timing performance. If necessary, you can turn off the **Quartus Prime Global Signal** logic option to manually prevent the use of global signals. The **Global & Other Fast Signals** table in the Compilation Report reports Global latch enables.

4.6 General Coding Guidelines

This section describes how coding styles impact synthesis of HDL code into the target Intel FPGA devices. You can improve your design efficiency and performance by following these recommended coding styles, and designing logic structures to match the appropriate device architecture.

4.6.1 Tri-State Signals

Use tri-state signals only when they are attached to top-level bidirectional or output pins.

Avoid lower-level bidirectional pins. Also avoid using the Z logic value unless it is driving an output or bidirectional pin. Even though some synthesis tools implement designs with internal tri-state signals correctly in Intel FPGA devices using multiplexer logic, do not use this coding style for Intel FPGA designs.

Note: In hierarchical block-based design flows, a hierarchical boundary cannot contain any bidirectional ports, unless the lower-level bidirectional port is connected directly through the hierarchy to a top-level output pin without connecting to any other design logic. If you use boundary tri-states in a lower-level block, synthesis software must push the tri-states through the hierarchy to the top level to make use of the tri-state drivers on output pins of Intel FPGA devices. Because pushing tri-states requires optimizing through hierarchies, lower-level tri-states are restricted with block-based design methodologies.

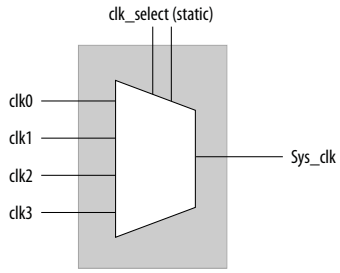
4.6.2 Clock Multiplexing

Clock multiplexing is sometimes used to operate the same logic function with different clock sources. This type of logic can introduce glitches that create functional problems. The delay inherent in the combinational logic can also lead to timing problems. Clock multiplexers trigger warnings from a wide range of design rule check and timing analysis tools.

Use dedicated hardware to perform clock multiplexing when it is available, instead of using multiplexing logic. For example, you can use the Clock Switchover feature or the Clock Control Block available in certain Intel FPGA devices. These dedicated hardware blocks avoid glitches, ensure that you use global low-skew routing lines, and avoid any possible hold time problems on the device due to logic delay on the clock line. Intel FPGA devices also support dynamic PLL reconfiguration, which is the safest and most robust method of changing clock rates during device operation.

If your design has too many clocks to use the clock control block, or if dynamic reconfiguration is too complex for your design, you can implement a clock multiplexer in logic cells. However, if you use this implementation, consider simultaneous toggling inputs and ensure glitch-free transitions.

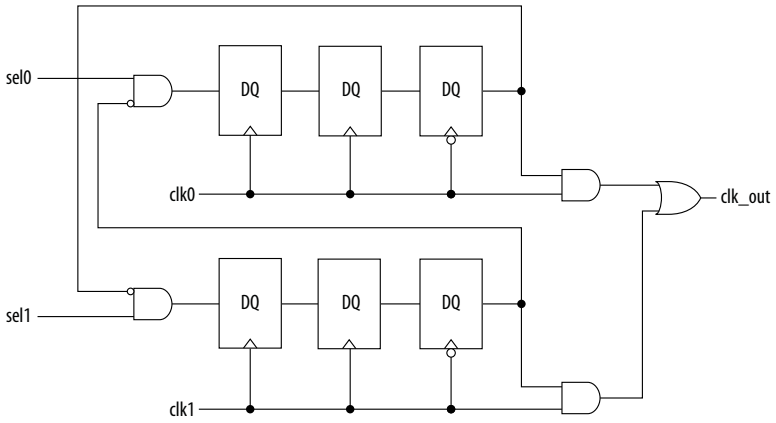
Figure 34. Simple Clock Multiplexer in a 6-Input LUT



Each device datasheet describes how LUT outputs can glitch during a simultaneous toggle of input signals, independent of the LUT function. Even though the 4:1 MUX function does not generate detectable glitches during simultaneous data input toggles, some cell implementations of multiplexing logic exhibit significant glitches, so this clock mux structure is not recommended. An additional problem with this implementation is that the output behaves erratically during a change in the `clk_select` signals. This behavior could create timing violations on all registers fed by the system clock and result in possible metastability.

A more sophisticated clock select structure can eliminate the simultaneous toggle and switching problems.

Figure 35. Glitch-Free Clock Multiplexer Structure



You can generalize this structure for any number of clock channels. The design ensures that no clock activates until all others are inactive for at least a few cycles, and that activation occurs while the clock is low. The design applies a `synthesis_keep` directive to the AND gates on the right side, which ensures there are no simultaneous toggles on the input of the `clk_out` OR gate.

Note: Switching from clock A to clock B requires that clock A continue to operate for at least a few cycles. If clock A stops immediately, the design sticks. The select signals are implemented as a "one-hot" control in this example, but you can use other encoding if you prefer. The input side logic is asynchronous and is not critical. This design can tolerate extreme glitching during the switch process.

Example 48. Verilog HDL Clock Multiplexing Design to Avoid Glitches

This example works with Verilog-2001.

```
module clock_mux (clk,clk_select,clk_out);

    parameter num_clocks = 4;

    input [num_clocks-1:0] clk;
    input [num_clocks-1:0] clk_select; // one hot
    output clk_out;

    genvar i;

    reg [num_clocks-1:0] ena_r0;
    reg [num_clocks-1:0] ena_r1;
    reg [num_clocks-1:0] ena_r2;
    wire [num_clocks-1:0] qualified_sel;

    // A look-up-table (LUT) can glitch when multiple inputs
    // change simultaneously. Use the keep attribute to
    // insert a hard logic cell buffer and prevent
    // the unrelated clocks from appearing on the same LUT.

    wire [num_clocks-1:0] gated_clks /* synthesis keep */;

    initial begin
        ena_r0 = 0;
        ena_r1 = 0;
        ena_r2 = 0;
    end

    generate
        for (i=0; i<num_clocks; i=i+1)
        begin : lp0
            wire [num_clocks-1:0] tmp_mask;
            assign tmp_mask = {num_clocks{1'b1}} ^ (1 << i);

            assign qualified_sel[i] = clk_select[i] & (~|(ena_r2 & tmp_mask));

            always @(posedge clk[i]) begin
                ena_r0[i] <= qualified_sel[i];
                ena_r1[i] <= ena_r0[i];
            end

            always @(negedge clk[i]) begin
                ena_r2[i] <= ena_r1[i];
            end

            assign gated_clks[i] = clk[i] & ena_r2[i];
        end
    endgenerate

    // These will not exhibit simultaneous toggle by construction
    assign clk_out = |gated_clks;

endmodule
```

Related Links

Intel FPGA IP Core Literature

4.6.3 Adder Trees

Structuring adder trees appropriately to match your targeted Intel FPGA device architecture can provide significant improvements in your design's efficiency and performance.

A good example of an application using a large adder tree is a finite impulse response (FIR) correlator. Using a pipelined binary or ternary adder tree appropriately can greatly improve the quality of your results.

4.6.3.1 Architectures with 6-Input LUTs in Adaptive Logic Modules

In Intel FPGA device families with 6-input LUT in their basic logic structure, ALMs can simultaneously add three bits. Take advantage of this feature by restructuring your code for better performance.

Although code targeting 4-input LUT architectures compiles successfully for 6-input LUT devices, the implementation can be inefficient. For example, to take advantage of the 6-input adaptive ALUT, you must rewrite large pipelined binary adder trees designed for 4-input LUT architectures. By restructuring the tree as a ternary tree, the design becomes much more efficient, significantly improving density utilization.

Example 49. Verilog HDL Pipelined Ternary Tree

The example shows a pipelined adder, but partitioning your addition operations can help you achieve better results in non-pipelined adders as well. If your design is not pipelined, a ternary tree provides much better performance than a binary tree. For example, depending on your synthesis tool, the HDL code `sum = (A + B + C) + (D + E)` is more likely to create the optimal implementation of a 3-input adder for `A + B + C` followed by a 3-input adder for `sum1 + D + E` than the code without the parentheses. If you do not add the parentheses, the synthesis tool may partition the addition in a way that is not optimal for the architecture.

```
module ternary_adder_tree (a, b, c, d, e, clk, out);
    parameter width = 16;
    input [width-1:0] a, b, c, d, e;
    input    clk;
    output [width-1:0] out;

    wire [width-1:0] sum1, sum2;
    reg [width-1:0] sumreg1, sumreg2;
    // registers

    always @ (posedge clk)
        begin
            sumreg1 <= sum1;
            sumreg2 <= sum2;
        end

    // 3-bit additions
    assign sum1 = a + b + c;
    assign sum2 = sumreg1 + d + e;
    assign out = sumreg2;
endmodule
```

4.6.4 State Machine HDL Guidelines

Synthesis tools can recognize and encode Verilog HDL and VHDL state machines during synthesis. This section presents guidelines to secure the best results when you use state machines.

When a synthesis tool recognizes a piece of code as a state machine, it can implement techniques that improve the design area and performance. For example, the tool can recode the state variables to improve the quality of results, or use the known properties of state machines to optimize other parts of the design.

To achieve the best results, synthesis tools often use one-hot encoding for FPGA devices and minimal-bit encoding for CPLD devices, although the choice of implementation can vary for different state machines and different devices. Refer to your synthesis tool documentation for specific ways to control the manner in which state machines are encoded.

To ensure proper recognition and inference of state machines and to improve the quality of results, observe the following guidelines for both Verilog HDL and VHDL:

- Assign default values to outputs derived from the state machine so that synthesis does not generate unwanted latches.
- Separate the state machine logic from all arithmetic functions and datapaths, including assigning output values.
- If your design contains an operation that more than one state uses, define the operation outside the state machine and cause the output logic of the state machine to use this value.
- Use a simple asynchronous or synchronous `reset` to ensure a defined power-up state. If your state machine design contains more elaborate `reset` logic, such as both an asynchronous `reset` and an asynchronous load, the Quartus Prime software generates regular logic rather than inferring a state machine.

If a state machine enters an illegal state due to a problem with the device, the design likely ceases to function correctly until the next `reset` of the state machine. Synthesis tools do not provide for this situation by default. The same issue applies to any other registers if there is some fault in the system. A `default` or `when others` clause does not affect this operation, assuming that your design never deliberately enters this state. Synthesis tools remove any logic generated by a default state if it is not reachable by normal state machine operation.

Many synthesis tools (including Quartus Prime synthesis) have an option to implement a safe state machine. The Quartus Prime software inserts extra logic to detect an illegal state and force the state machine's transition to the `reset` state. It is commonly used when the state machine can enter an illegal state. The most common cause of this situation is a state machine that has control inputs that come from another clock domain, such as the control logic for a dual-clock FIFO.

This option protects only state machines by forcing them into the `reset` state. All other registers in the design are not protected this way. If the design has asynchronous inputs, Intel recommends using a synchronization register chain instead of relying on the safe state machine option.

4.6.4.1 Verilog HDL State Machines

To ensure proper recognition and inference of Verilog HDL state machines, observe the following additional Verilog HDL guidelines.

Refer to your synthesis tool documentation for specific coding recommendations. If the synthesis tool doesn't recognize and infer the state machine, the tool implements the state machine as regular logic gates and registers, and the state machine doesn't appear as a state machine in the **Analysis & Synthesis** section of the Quartus Prime Compilation Report. In this case, Quartus Prime synthesis does not perform any optimizations specific to state machines.

- If you are using the SystemVerilog standard, use enumerated types to describe state machines.
- Represent the states in a state machine with the parameter data types in Verilog-1995 and Verilog-2001, and use the parameters to make state assignments. This parameter implementation makes the state machine easier to read and reduces the risk of errors during coding.
- Do not directly use integer values for state variables, such as `next_state <= 0`. However, using an integer does not prevent inference in the Quartus Prime software.
- Quartus Prime software doesn't infer a state machine if the state transition logic uses arithmetic similar to the following example:

```
case (state)
    0: begin
        if (ena) next_state <= state + 2;
        else next_state <= state + 1;
        end
    1: begin
    ...
endcase
```

- Quartus Prime software doesn't infer a state machine if the state variable is an output.
- Quartus Prime software doesn't infer a state machine for signed variables.

4.6.4.1.1 Verilog-2001 State Machine Coding Example

The following module `verilog_fsm` is an example of a typical Verilog HDL state machine implementation. This state machine has five states.

The asynchronous reset sets the variable `state` to `state_0`. The sum of `in_1` and `in_2` is an output of the state machine in `state_1` and `state_2`. The difference (`in_1` – `in_2`) is also used in `state_1` and `state_2`. The temporary variables `tmp_out_0` and `tmp_out_1` store the sum and the difference of `in_1` and `in_2`. Using these temporary variables in the various states of the state machine ensures proper resource sharing between the mutually exclusive states.

Example 50. Verilog-2001 State Machine

```
module verilog_fsm (clk, reset, in_1, in_2, out);
    input clk, reset;
    input [3:0] in_1, in_2;
    output [4:0] out;
    parameter state_0 = 3'b000;
    parameter state_1 = 3'b001;
```

```
    parameter state_2 = 3'b010;
    parameter state_3 = 3'b011;
    parameter state_4 = 3'b100;

    reg [4:0] tmp_out_0, tmp_out_1, tmp_out_2;
    reg [2:0] state, next_state;

    always @ (posedge clk or posedge reset)
    begin
        if (reset)
            state <= state_0;
        else
            state <= next_state;
    end
    always @ (*)
    begin
        tmp_out_0 = in_1 + in_2;
        tmp_out_1 = in_1 - in_2;
        case (state)
            state_0: begin
               tmp_out_2 = in_1 + 5'b00001;
               next_state = state_1;
            end
            state_1: begin
                if (in_1 < in_2) begin
                    next_state = state_2;
                    tmp_out_2 = tmp_out_0;
                end
                else begin
                    next_state = state_3;
                    tmp_out_2 = tmp_out_1;
                end
            end
            state_2: begin
                tmp_out_2 = tmp_out_0 - 5'b00001;
                next_state = state_3;
            end
            state_3: begin
                tmp_out_2 = tmp_out_1 + 5'b00001;
                next_state = state_0;
            end
            state_4:begin
                tmp_out_2 = in_2 + 5'b00001;
                next_state = state_0;
            end
            default:begin
                tmp_out_2 = 5'b00000;
                next_state = state_0;
            end
        endcase
    end
    assign out = tmp_out_2;
endmodule
```

You can achieve an equivalent implementation of this state machine by using `‘define` instead of the `parameter` data type, as follows:

```
‘define state_0 3'b000
‘define state_1 3'b001
‘define state_2 3'b010
‘define state_3 3'b011
‘define state_4 3'b100
```

In this case, you assign `` `state_x `` instead of `state_x` to `state` and `next_state`, for example:

```
next_state <= ‘state_3;
```

Note: Although Intel supports the `‘define` construct, use the `parameter` data type, because it preserves the state names throughout synthesis.

4.6.4.1.2 SystemVerilog State Machine Coding Example

Use the following coding style to describe state machines in SystemVerilog.

Example 51. SystemVerilog State Machine Using Enumerated Types

The module `enum_fsm` is an example of a SystemVerilog state machine implementation that uses enumerated types.

In Quartus Prime Pro Edition synthesis, the enumerated type that defines the states for the state machine must be of an unsigned integer type. If you do not specify the enumerated type as `int unsigned`, synthesis uses a signed `int` type by default. In this case, the Quartus Prime software synthesizes the design, but does not infer or optimize the logic as a state machine.

```
module enum_fsm (input clk, reset, input int data[3:0], output int o);
enum int unsigned { S0 = 0, S1 = 2, S2 = 4, S3 = 8 } state, next_state;
always_comb begin : next_state_logic
      next_state = S0;
      case(state)
        S0: next_state = S1;
        S1: next_state = S2;
        S2: next_state = S3;
        S3: next_state = S3;
      endcase
end
always_comb begin
      case(state)
         S0: o = data[3];
         S1: o = data[2];
         S2: o = data[1];
         S3: o = data[0];
      endcase
end
always_ff@(posedge clk or negedge reset) begin
      if(~reset)
         state <= S0;
      else
         state <= next_state;
end
endmodule
```

4.6.4.2 VHDL State Machines

To ensure proper recognition and inference of VHDL state machines, represent the different states with enumerated types, and use the corresponding types to make state assignments.

This implementation makes the state machine easier to read, and reduces the risk of errors during coding. If your RTL does not represent states with an enumerated type, Quartus Prime synthesis (and other synthesis tools) do not recognize the state machine. Instead, synthesis implements the state machine as regular logic gates and registers. Consequently, and the state machine does not appear in the state machine list of the Quartus Prime Compilation Report, **Analysis & Synthesis** section. Moreover, Quartus Prime synthesis does not perform any of the optimizations that are specific to state machines.

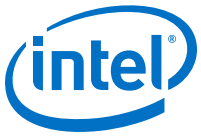
4.6.4.2.1 VHDL State Machine Coding Example

The following state machine has five states. The asynchronous reset sets the variable `state` to `state_0`.

The sum of `in1` and `in2` is an output of the state machine in `state_1` and `state_2`. The difference (`in1 - in2`) is also used in `state_1` and `state_2`. The temporary variables `tmp_out_0` and `tmp_out_1` store the sum and the difference of `in1` and `in2`. Using these temporary variables in the various states of the state machine ensures proper resource sharing between the mutually exclusive states.

Example 52. VHDL State Machine

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
ENTITY vhdl_fsm IS
   PORT(
      clk: IN STD_LOGIC;
      reset: IN STD_LOGIC;
      in1: IN UNSIGNED(4 downto 0);
      in2: IN UNSIGNED(4 downto 0);
      out_1: OUT UNSIGNED(4 downto 0)
      );
END vhdl_fsm;
ARCHITECTURE rtl OF vhdl_fsm IS
   TYPE Tstate IS (state_0, state_1, state_2, state_3, state_4);
   SIGNAL state: Tstate;
   SIGNAL next_state: Tstate;
BEGIN
   PROCESS(clk, reset)
   BEGIN
      IF reset = '1' THEN
            state <=state_0;
      ELSIF rising_edge(clk) THEN
            state <= next_state;
      END IF;
   END PROCESS;
PROCESS (state, in1, in2)
      VARIABLE tmp_out_0: UNSIGNED (4 downto 0);
      VARIABLE tmp_out_1: UNSIGNED (4 downto 0);
   BEGIN
      tmp_out_0 := in1 + in2;
      tmp_out_1 := in1 - in2;
      CASE state IS
         WHEN state_0 =>
            out_1 <= in1;
            next_state <= state_1;
         WHEN state_1 =>
            IF (in1 < in2) then
               next_state <= state_2;
               out_1 <= tmp_out_0;
            ELSE
               next_state <= state_3;
               out_1 <= tmp_out_1;
            END IF;
         WHEN state_2 =>
            IF (in1 < "0100") then
               out_1 <= tmp_out_0;
            ELSE
               out_1 <= tmp_out_1;
            END IF;
               next_state <= state_3;
         WHEN state_3 =>
               out_1 <= "11111";
               next_state <= state_4;
```



```
         WHEN state_4 =>
               out_1 <= in2;
               next_state <= state_0;
         WHEN OTHERS =>
               out_1 <= "00000";
               next_state <= state_0;
      END CASE;
   END PROCESS;
END rtl;
```

4.6.5 Multiplexer HDL Guidelines

Multiplexers form a large portion of the logic utilization in many FPGA designs. By optimizing your multiplexer logic, you ensure the most efficient implementation.

This section addresses common problems and provides design guidelines to achieve optimal resource utilization for multiplexer designs. The section also describes various types of multiplexers, and how they are implemented.

For more information, refer to the *Advanced Synthesis Cookbook*.

4.6.5.1 Quartus Prime Software Option for Multiplexer Restructuring

Quartus Prime Pro Edition synthesis provides the **Restructure Multiplexers** logic option that extracts and optimizes buses of multiplexers during synthesis. The default **Auto** for this option setting uses the optimization whenever beneficial for your design. You can turn the option on or off specifically to have more control over use.

Even with this Quartus Prime-specific option turned on, it is beneficial to understand how your coding style can be interpreted by your synthesis tool, and avoid the situations that can cause problems in your design.

4.6.5.2 Multiplexer Types

This section addresses how Quartus Prime synthesis creates multiplexers from various types of HDL code.

State machines, CASE statements, and IF statements are all common sources of multiplexer logic in designs. These HDL structures create different types of multiplexers, including binary multiplexers, selector multiplexers, and priority multiplexers.

The first step toward optimizing multiplexer structures for best results is to understand how Quartus Prime infers and implements multiplexers from HDL code.

4.6.5.2.1 Binary Multiplexers

Binary multiplexers select inputs based on binary-encoded selection bits.

Device families featuring 6-input look up tables (LUTs) are perfectly suited for 4:1 multiplexer building blocks (4 data and 2 select inputs). The extended input mode facilitates implementing 8:1 blocks, and the fractured mode handles residual 2:1 multiplexer pairs.

Example 53. Verilog HDL Binary-Encoded Multiplexers

```
case (sel)
   2'b00: z = a;
   2'b01: z = b;
   2'b10: z = c;
   2'b11: z = d;
endcase
```

4.6.5.2.2 Selector Multiplexers

Selector multiplexers have a separate select line for each data input. The select lines for the multiplexer are one-hot encoded. Quartus Prime commonly builds selector multiplexers as a tree of AND and OR gates.

Even though the implementation of a tree-shaped, N-input selector multiplexer is slightly less efficient than a binary multiplexer, in many cases the select signal is the output of a decoder. Quartus Prime synthesis combines the selector and decoder into a binary multiplexer.

Example 54. Verilog HDL One-Hot-Encoded CASE Statement

```
case (sel)
   4'b0001: z = a;
   4'b0010: z = b;
   4'b0100: z = c;
   4'b1000: z = d;
   default: z = 1'bx;
endcase
```

4.6.5.2.3 Priority Multiplexers

In priority multiplexers, the select logic implies a priority. The options to select the correct item must be checked in a specific order based on signal priority.

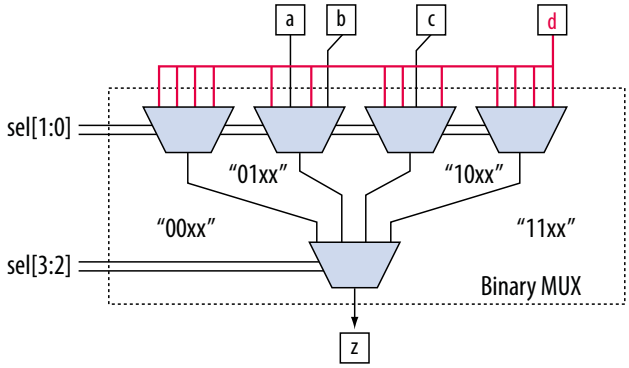
Synthesis tools commonlyinfer these structures from `IF`, `ELSE`, `WHEN`, `SELECT`, and `?:` statements in VHDL or Verilog HDL.

Example 55. VHDL IF Statement Implying Priority

The multiplexers form a chain, evaluating each condition or select bit sequentially.

```
IF cond1 THEN z <= a;
ELSIF cond2 THEN z <= b;
ELSIF cond3 THEN z <= c;
ELSE z <= d;
END IF;
```

Figure 36. Priority Multiplexer Implementation of an IF Statement



Depending on the number of multiplexers in the chain, the timing delay through this chain can become large, especially for device families with 4-input LUTs.

To improve the timing delay through the multiplexer, avoid priority multiplexers if priority is not required. If the order of the choices is not important to the design, use a CASE statement to implement a binary or selector multiplexer instead of a priority multiplexer. If delay through the structure is important in a multiplexed design requiring priority, consider recoding the design to reduce the number of logic levels to minimize delay, especially along your critical paths.

4.6.5.3 Implicit Defaults in IF Statements

The IF statements in Verilog HDL and VHDL can be a convenient way to specify conditions that do not easily lend themselves to a CASE-type approach.

However, using IF statements can result in complicated multiplexer trees that are not easy for synthesis tools to optimize. In particular, every IF statement has an implicit ELSE condition, even when it is not specified. These implicit defaults can cause additional complexity in a multiplexed design.

There are several ways you can simplify multiplexed logic and remove unneeded defaults. The optimal method may be to recode the design so the logic takes the structure of a 4:1 CASE statement. Alternatively, if priority is important, you can restructure the code to reduce default cases and flatten the multiplexer. Examine whether the default "ELSE IF" conditions are don't care cases. You may be able to create a default ELSE statement to make the behavior explicit. Avoid unnecessary default conditions in your multiplexer logic to reduce the complexity and logic utilization required to implement your design.

4.6.5.4 `default` or OTHERS CASE Assignment

To fully specify the cases in a CASE statement, include a `default` (Verilog HDL) or OTHERS (VHDL) assignment.

This assignment is especially important in one-hot encoding schemes where many combinations of the select lines are unused. Specifying a case for the unused select line combinations gives the synthesis tool information about how to synthesize these cases, and is required by the Verilog HDL and VHDL language specifications.

For some designs you do not need to consider the outcome in the unused cases, because these cases are unreachable. For these types of designs, you can specify any value for the `default` or OTHERS assignment. However, the assignment value you choose can have a large effect on the logic utilization required to implement the design.

To obtain best results, explicitly define invalid CASE selections with a separate `default` or OTHERS statement, instead of combining the invalid cases with one of the defined cases.

If the value in the invalid cases is not important, specify those cases explicitly by assigning the X (don't care) logic value instead of choosing another value. This assignment allows your synthesis tool to perform the best area optimizations.

4.6.6 Cyclic Redundancy Check Functions

CRC computations are used heavily by communications protocols and storage devices to detect any corruption of data. These functions are highly effective; there is a very low probability that corrupted data can pass a 32-bit CRC check

CRC functions typically use wide XOR gates to compare the data. The way synthesis tools flatten and factor these XOR gates to implement the logic in FPGA LUTs can greatly impact the area and performance results for the design. XOR gates have a cancellation property that creates an exceptionally large number of reasonable factoring combinations, so synthesis tools cannot always choose the best result by default.

The 6-input ALUT has a significant advantage over 4-input LUTs for these designs. When properly synthesized, CRC processing designs can run at high speeds in devices with 6-input ALUTs.

The following guidelines help you improve the quality of results for CRC designs in Intel FPGA devices.

4.6.6.1 If Performance is Important, Optimize for Speed

To minimize area and depth of levels of logic, synthesis tools flatten XOR gates.

By default, Quartus Prime Pro Edition synthesis targets area optimization for XOR gates. Therefore, for more focus on depth reduction, set the synthesis optimization technique to speed.

Note: Flattening for depth sometimes causes a significant increase in area.

4.6.6.2 Use Separate CRC Blocks Instead of Cascaded Stages

Some designs optimize CRC to use cascaded stages (for example, four stages of 8 bits). In such designs, Quartus Prime synthesis uses intermediate calculations (such as the calculations after 8, 24, or 32 bits) depending on the data width.

This design is not optimal for FPGA devices. The XOR cancellations that Quartus Prime synthesis performs in CRC designs mean that the function does not require all the intermediate calculations to determine the final result. Therefore, forcing the use of intermediate calculations increases the area required to implement the function, as

well as increasing the logic depth because of the cascading. It is typically better to create full separate CRC blocks for each data width that you require in the design, and then multiplex them together to choose the appropriate mode at a given time

4.6.6.3 Use Separate CRC Blocks Instead of Allowing Blocks to Merge

Synthesis tools often attempt to optimize CRC designs by sharing resources and extracting duplicates in two different CRC blocks because of the factoring options in the XOR logic.

The CRC logic allows significant reductions, but this works best when each CRC function is optimized separately. Check for duplicate extraction behavior if you have different CRC functions that are driven by common data signals or that feed the same destination signals.

If you are having problems with the quality of results and you see that two CRC functions are sharing logic, ensure that the blocks are synthesized independently using one of the following methods:

- Define each CRC block as a separate design partition in an hierarchical compilation design flow.
- Synthesize each CRC block as a separate project in your third-party synthesis tool and then write a separate Verilog Quartus Mapping (**.vqm**) or EDIF netlist file for each.

4.6.6.4 Take Advantage of Latency if Available

If your design can use more than one cycle to implement the CRC functionality, adding registers and retiming the design can help reduce area, improve performance, and reduce power utilization.

If your synthesis tool offers a retiming feature (such as the Quartus Prime software **Perform gate-level register retiming** option), you can insert an extra bank of registers at the input and allow the retiming feature to move the registers for better results. You can also build the CRC unit half as wide and alternate between halves of the data in each clock cycle.

4.6.6.5 Save Power by Disabling CRC Blocks When Not in Use

CRC designs are heavy consumers of dynamic power because the logic toggles whenever there is a change in the design.

To save power, use clock enables to disable the CRC function for every clock cycle that the logic is not required. Some designs don't check the CRC results for a few clock cycles while other logic is performing. It is valuable to disable the CRC function even for this short amount of time.

4.6.6.6 Use the Device Synchronous Load (sload) Signal to Initialize

The data in many CRC designs must be initialized to 1's before operation. If your target device supports the use of the `sload` signal, use it to set all the registers in your design to 1's before operation.

To enable use of the `sload` signal, follow the coding guidelines in this chapter. You can check the register equations in the Chip Planner to ensure that the signal was used as expected.

If you must force a register implementation using an `sload` signal, refer to*Designing with Low-Level Primitives User Guide* to see how you can use low-level device primitives.

Related Links

- Secondary Register Control Signals Such as Clear and Clock Enable on page 131
  The registers in Intel FPGAs provide a number of secondary control signals. Use these signals to implement control logic for each register without using extra logic cells.
- Designing with Low-Level Primitives User Guide

4.6.7 Comparator HDL Guidelines

This section provides information about the different types of implementations available for comparators (<, >, or ==), and provides suggestions on how you can code your design to encourage a specific implementation. Synthesis tools, including Quartus Prime Pro Edition synthesis, use device and context-specific implementation rules, and select the best one for your design.

Synthesis tools implement the == comparator in general logic cells. Additionally, synthesis tools implement the < comparison either using the carry chain or general logic cells. In devices with 6-input ALUTs, the carry chain is capable of comparing up to three bits per cell. The carry chain implementation tends to be faster than the general logic on standalone benchmark test cases, but can result in lower performance when it is part of a larger design due to the increased restriction on the Fitter. The area requirement is similar for most input patterns. The synthesis tools select an appropriate implementation based on the input pattern.

If you are using Quartus Prime synthesis, you can guide the tool by using specific coding styles. To select a carry chain implementation explicitly, rephrase your comparison in terms of addition. As a simple example, the following coding style allows the synthesis tool to select the implementation, which is most likely using general logic cells in modern device families:

```
wire [6:0] a,b;
wire alb = a<b;
```

In the following coding style, the synthesis tool uses a carry chain (except for a few cases, such as when the chain is very short or the signals `a` and `b` minimize to the same signal):

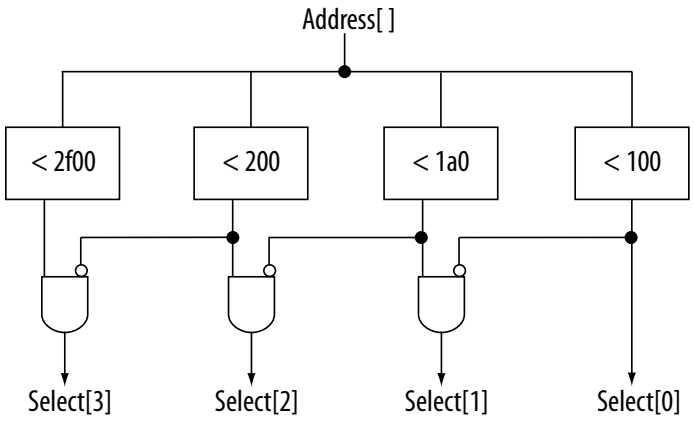
```
wire [6:0] a,b;
wire [7:0] tmp = a - b;
wire alb = tmp[7]
```

This second coding style uses the top bit of the `tmp` signal, which is 1 in twos complement logic if *a* is less than *b*, because the subtraction *a* – *b* results in a negative number.

If you have any information about the range of the input, you have "don't care" values that you can use to optimize the design. Because this information is not available to the synthesis tool, you can often reduce the device area required to implement the comparator with specific hand implementation of the logic.

You can also check whether a bus value is within a constant range with a small amount of logic area by using the following logic structure . This type of logic occurs frequently in address decoders.

Figure 37. Example Logic Structure for Using Comparators to Check a Bus Value Range



4.6.8 Counter HDL Guidelines

Implementing counters in HDL code is easy; they are implemented with an adder followed by registers.

Register control signals, such as enable (`ena`), synchronous clear (`sclr`), and synchronous load (`sload`), are available. For the best area utilization, ensure that the up/down control or controls are expressed in terms of one addition instead of two separate addition operators.

If you use the following coding style, your synthesis tool may implement two separate carry chains for addition (if it doesn't detect the issue and optimize the logic):

```
out <= count_up ? out + 1 : out - 1;
```

The following coding style requires only one adder along with some other logic:

```
out <= out + (count_up ? 1 : -1);
```

In this case, the coding style better matches the device hardware because there is only one carry chain adder, and the –1 constant logic is implemented in the LUT in front of the adder without adding extra area utilization.

4.7 Designing with Low-Level Primitives

Low-level HDL design is the practice of using low-level primitives and assignments to dictate a particular hardware implementation for a piece of logic. Low-level primitives are small architectural building blocks that assist you in creating your design.

With the Quartus Prime software, you can use low-level HDL design techniques to force a specific hardware implementation that can help you achieve better resource utilization or faster timing results.

Note: Using low-level primitives is an optional advanced technique to help with specific design challenges. For many designs, synthesizing generic HDL source code and Intel FPGA IP cores give you the best results.

Low-level primitives allow you to use the following types of coding techniques:

- Instantiate the logic cell or LCELL primitive to prevent Quartus Prime Pro Edition synthesis from performing optimizations across a logic cell
- Create carry and cascade chains using CARRY, CARRY_SUM, and CASCADE primitives
- Instantiate registers with specific control signals using DFF primitives
- Specify the creation of LUT functions by identifying the LUT boundaries
- Use I/O buffers to specify I/O standards, current strengths, and other I/O assignments
- Use I/O buffers to specify differential pin names in your HDL code, instead of using the automatically-generated negative pin name for each pair

For details about and examples of using these types of assignments, refer to the *Designing with Low-Level Primitives User Guide*.

Related Links

Designing with Low-Level Primitives User Guide

4.8 Document Revision History

The following revisions history applies to this chapter.

Table 28. Document Revision History

Date	Version	Changes
2017.05.08	17.0.0	• Updated example: Verilog HDL Multiply-Accumulator • Updated information about use of safe state machine. • Revised Check Read-During-Write Behavior. • Revised Controlling RAM Inference and Implementation. • Revised Single-Clock Synchronous RAM with Old Data Read-During-Write Behavior. • Revised Single-Clock Synchronous RAM with New Data Read-During-Write Behavior. • Updated and moved template for VHDL Single-Clock Simple Dual Port Synchronous RAM with New Data Read-During-Write Behavior. • Revised Inferring ROM Functions from HDL Code. • Removed example: VHDL 8-Bit Wide, 64-Bit Long Shift Register with Evenly Spaced Taps. • Removed example: Verilog HDL D-Type Flipflop (Register) With ena, aclr, and aload Control Signals • Removed example: VHDL D-Type Flipflop (Register) With ena, aclr, and aload Control Signals • Added example: Verilog D-type Flipflop bus with Secondary Signals

continued...

Date	Version	Changes
		• Removed references to 4-input LUT-based devices. • Removed references to Integrated Synthesis. • Created example: Avoid this VHDL Coding Style.
2016.10.31	16.1.0	• Provided corrected Verilog HDL Pipelined Binary Tree and Ternary Tree examples. • Implemented Intel rebranding.
2016.05.03	16.0.0	• Added information about use of safe state machine. • Updated example code templates with latest coding styles.
2015.11.02	15.1.0	• Changed instances of *Quartus II* to *Quartus Prime*.
2015.05.04	15.0.0	Added information and reference about ramstyle attribute for sift register inference.
2014.12.15	14.1.0	Updated location of Fitter Settings, Analysis & Synthesis Settings, and Physical Optimization Settings to Compiler Settings.
2014.08.18	14.0.a10.0	• Added recommendation to use register pipelining to obtain high performance in DSP designs.
2014.06.30	14.0.0	Removed obsolete MegaWizard Plug-In Manager support.
November 2013	13.1.0	Removed HardCopy device support.
June 2012	12.0.0	• Revised section on inserting Altera templates. • Code update for Example 11-51. • Minor corrections and updates.
November 2011	11.1.0	• Updated document template. • Minor updates and corrections.
December 2010	10.1.0	• Changed to new document template. • Updated Unintentional Latch Generation content. • Code update for Example 11-18.
July 2010	10.0.0	• Added support for mixed-width RAM • Updated support for no_rw_check for inferring RAM blocks • Added support for byte-enable
November 2009	9.1.0	• Updated support for Controlling Inference and Implementation in Device RAM Blocks • Updated support for Shift Registers
March 2009	9.0.0	• Corrected and updated several examples • Added support for Arria II GX devices • Other minor changes to chapter
November 2008	8.1.0	Changed to 8-1/2 x 11 page size. No change to content.
May 2008	8.0.0	Updates for the Quartus Prime software version 8.0 release, including: • Added information to "RAM • Functions—Inferring ALTSYNCRAM and ALTDPRAM Megafunctions from HDL Code" on page 6–13 • Added information to "Avoid Unsupported Reset and Control Conditions" on page 6–14 • Added information to "Check Read-During-Write Behavior" on page 6–16 • Added two new examples to "ROM Functions—Inferring ALTSYNCRAM and LPM_ROM Megafunctions from HDL Code" on page 6–28: Example 6–24 and Example 6–25 • Added new section: "Clock Multiplexing" on page 6–46 • Added hyperlinks to references within the chapter • Minor editorial updates

Related Links

Altera Documentation Archive

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

5 Recommended Design Practices

This chapter provides design recommendations for Intel FPGA devices.

Current FPGA applications have reached the complexity and performance requirements of ASICs. In the development of complex system designs, design practices have an enormous impact on the timing performance, logic utilization, and system reliability of a device. Well-coded designs behave in a predictable and reliable manner even when retargeted to different families or speed grades. Good design practices also aid in successful design migration between FPGA and ASIC implementations for prototyping and production.

For optimal performance, reliability, and faster time-to-market when designing with Intel FPGA devices, you should adhere to the following guidelines:

- Understand the impact of synchronous design practices
- Follow recommended design techniques, including hierarchical design partitioning, and timing closure guidelines
- Take advantage of the architectural features in the targeted device

5.1 Following Synchronous FPGA Design Practices

The first step in good design methodology is to understand the implications of your design practices and techniques. This section outlines the benefits of optimal synchronous design practices and the hazards involved in other approaches.

Good synchronous design practices can help you meet your design goals consistently. Problems with other design techniques can include reliance on propagation delays in a device, which can lead to race conditions, incomplete timing analysis, and possible glitches.

In a synchronous design, a clock signal triggers every event. As long as you ensure that all the timing requirements of the registers are met, a synchronous design behaves in a predictable and reliable manner for all process, voltage, and temperature (PVT) conditions. You can easily migrate synchronous designs to different device families or speed grades.

5.1.1 Implementing Synchronous Designs

In a synchronous design, the clock signal controls the activities of all inputs and outputs.

On every active edge of the clock (usually the rising edge), the data inputs of registers are sampled and transferred to outputs. Following an active clock edge, the outputs of combinational logic feeding the data inputs of registers change values. This change triggers a period of instability due to propagation delays through the logic as the

signals go through several transitions and finally settle to new values. Changes that occur on data inputs of registers do not affect the values of their outputs until after the next active clock edge.

Because the internal circuitry of registers isolates data outputs from inputs, instability in the combinational logic does not affect the operation of the design as long as you meet the following timing requirements:

- Before an active clock edge, you must ensure that the data input has been stable for at least the setup time of the register.
- After an active clock edge, you must ensure that the data input remains stable for at least the hold time of the register.

  When you specify all of your clock frequencies and other timing requirements, the Quartus Prime TimeQuest Timing Analyzer reports actual hardware requirements for the setup times (t_{SU}) and hold times (t_H) for every pin in your design. By meeting these external pin requirements and following synchronous design techniques, you ensure that you satisfy the setup and hold times for all registers in your device.

  Tip: To meet setup and hold time requirements on all input pins, any inputs to combinational logic that feed a register should have a synchronous relationship with the clock of the register. If signals are asynchronous, you can register the signals at the inputs of the device to help prevent a violation of the required setup and hold times.

  When you violate the setup or hold time of a register, you might oscillate the output, or set the output to an intermediate voltage level between the high and low levels called a metastable state. In this unstable state, small perturbations such as noise in power rails can cause the register to assume either the high or low voltage level, resulting in an unpredictable valid state. Various undesirable effects can occur, including increased propagation delays and incorrect output states. In some cases, the output can even oscillate between the two valid states for a relatively long period of time.

5.1.2 Asynchronous Design Hazards

Some designers use asynchronous techniques such as ripple counters or pulse generators in programmable logic device (PLD) designs, enabling them to take "short cuts" to save device resources.

Asynchronous design techniques have inherent problems such as relying on propagation delays in a device, which can vary with temperature and voltage fluctuations, resulting in incomplete timing constraints and possible glitches and spikes.

Some asynchronous design structures depend on the relative propagation delays of signals to function correctly. In these cases, race conditions arise where the order of signal changes affect the output of the logic. Depending on how the design is placed and routed in the device, PLD designs can have varying timing delays with each compilation. Therefore, it is almost impossible to determine the timing delay associated with a particular block of logic ahead of time. As devices become faster due to process improvements, the delays in an asynchronous design may decrease, resulting in a design that does not function as expected. Relying on a particular delay also makes asynchronous designs difficult to migrate to different architectures, devices, or speed grades.

The timing of asynchronous design structures is often difficult or impossible to model with timing assignments and constraints. If you do not have complete or accurate timing constraints, the timing-driven algorithms used by your synthesis and place-and-route tools may not be able to perform the best optimizations, and the reported results may not be complete.

Some asynchronous design structures can generate harmful glitches, which are pulses that are very short compared to clock periods. Most glitches are generated by combinational logic. When the inputs to the combinational logic change, the outputs exhibit several glitches before they settle to their new values. These glitches can propagate through the combinational logic, leading to incorrect values on the outputs in asynchronous designs. In a synchronous design, glitches on the data inputs of registers are normal events that have no negative consequences because the data is not processed until the next clock edge.

5.2 HDL Design Guidelines

When designing with HDL code, you should understand how a synthesis tool interprets different HDL design techniques and what results to expect.

Your design style can affect logic utilization and timing performance, as well as the design's reliability. This section describes basic design techniques that ensure optimal synthesis results for designs targeted to Intel FPGA devices while avoiding several common causes of unreliability and instability. Intel recommends to design your combinational logic carefully to avoid potential problems. Pay attention to your clocking schemes so that you can maintain synchronous functionality and avoid timing problems.

5.2.1 Optimizing Combinational Logic

Combinational logic structures consist of logic functions that depend only on the current state of the inputs. In Intel FPGAs, these functions are implemented in the look-up tables (LUTs) with either logic elements (LEs) or adaptive logic modules (ALMs).

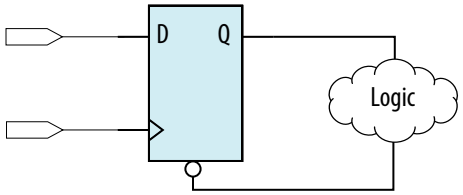
For cases where combinational logic feeds registers, the register control signals can implement part of the logic function to save LUT resources. By following the recommendations in this section, you can improve the reliability of your combinational design.

5.2.1.1 Avoid Combinational Loops

Combinational loops are among the most common causes of instability and unreliability in digital designs. Combinational loops generally violate synchronous design principles by establishing a direct feedback loop that contains no registers.

Avoid combinational loops whenever possible. In a synchronous design, feedback loops should include registers. For example, a combinational loop occurs when the left-hand side of an arithmetic expression also appears on the right-hand side in HDL code. A combinational loop also occurs when you feed back the output of a register to an asynchronous pin of the same register through combinational logic.

Figure 38. Combinational Loop Through Asynchronous Control Pin



Tip: Use recovery and removal analysis to perform timing analysis on asynchronous ports, such as `clear` or `reset` in the Quartus Prime software.

Combinational loops are inherently high-risk design structures for the following reasons:

- Combinational loop behavior generally depends on relative propagation delays through the logic involved in the loop. As discussed, propagation delays can change, which means the behavior of the loop is unpredictable.
- In many design tools, combinational loops can cause endless computation loops . Most tools break open combinational loops to process the design. The various tools used in the design flow may open a given loop differently, and process it in a way inconsistent with the original design intent.

5.2.1.2 Avoid Unintended Latch Inference

Avoid using latches to ensure that you can completely analyze the timing performance and reliability of your design. A latch is a small circuit with combinational feedback that holds a value until a new value is assigned. You can implement latches with the Quartus Prime Text Editor or Block Editor.

A common mistake in HDL code is unintended latch inference; Quartus Prime Synthesis issues a warning message if this occurs. Unlike other technologies, a latch in FPGA architecture is not significantly smaller than a register. However, the architecture is not optimized for latch implementation and latches generally have slower timing performance compared to equivalent registered circuitry.

Latches have a transparent mode in which data flows continuously from input to output. A positive latch is in transparent mode when the enable signal is high (low for a negative latch). In transparent mode, glitches on the input can pass through to the output because of the direct path created. This presents significant complexity for timing analysis. Typical latch schemes use multiple enable phases to prevent long transparent paths from occurring. However, timing analysis cannot identify these safe applications.

The TimeQuest analyzer analyzes latches as synchronous elements clocked on the falling edge of the positive latch signal by default. It allows you to treat latches as having nontransparent start and end points. Be aware that even an instantaneous transition through transparent mode can lead to glitch propagation. The TimeQuest analyzer cannot perform cycle-borrowing analysis.

Due to various timing complexities, latches have limited support in formal verification tools. Therefore, you should not rely on formal verification for a design that includes latches.

5.2.1.3 Avoid Delay Chains in Clock Paths

Delays in PLD designs can change with each placement and routing cycle. Effects such as rise and fall time differences and on-chip variation mean that delay chains, especially those placed on clock paths, can cause significant problems in your design. Avoid using delay chains to prevent these kinds of problems.

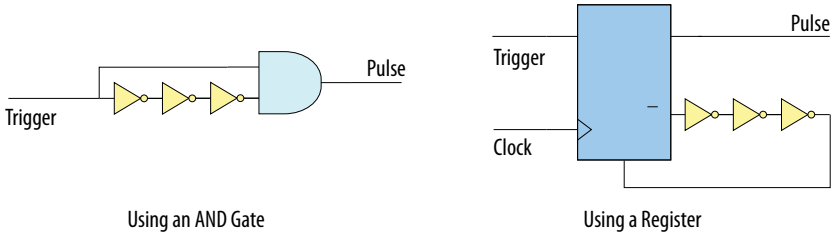
You require delay chains when you use two or more consecutive nodes with a single fan-in and a single fan-out to cause delay. Inverters are often chained together to add delay. Delay chains are sometimes used to resolve race conditions created by other asynchronous design practices.

In some ASIC designs, delays are used for buffering signals as they are routed around the device. This functionality is not required in FPGA devices because the routing structure provides buffers throughout the device.

5.2.1.4 Use Synchronous Pulse Generators

To design a pulse generator use synchronous techniques. The following figure shows two methods for asynchronous pulse generation. The first method uses a delay chain to generate a single pulse (pulse generator). The second method generates a series of pulses (multivibrators).

Figure 39. Asynchronous Pulse Generators



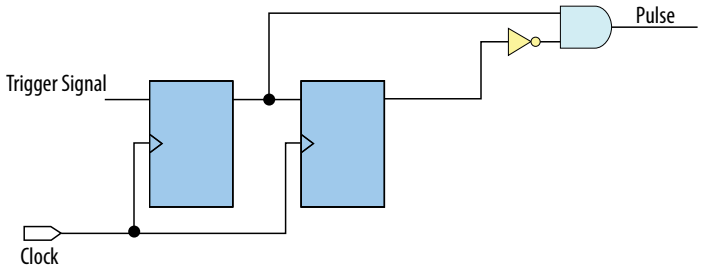
A trigger signal feeds both inputs of a 2-input AND gate, but the design adds inverters to create a delay chain to one of the inputs. The width of the pulse depends on the time differences between the path that feeds the gate directly, and the path that goes through the delay chain. This is the same mechanism responsible for the generation of glitches in combinational logic following a change of input values. This technique artificially increases the width of the glitch.

A register's output drives the same register's asynchronous reset signal through a delay chain. The register resets itself asynchronously after a certain delay.

The width of pulses generated in this way are difficult for synthesis and place-and-route to determine, set, or verify. The actual pulse width can only be determined after placement and routing, when routing and propagation delays are known. You cannot reliably create a specific pulse width when creating HDL code, and it cannot be set by EDA tools. The pulse may not be wide enough for the application under all PVT conditions. Also, the pulse width changes if you change to a different device. Additionally, verification is difficult because static timing analysis cannot verify the pulse width.

Multivibrators use a glitch generator to create pulses, together with a combinational loop that turns the circuit into an oscillator. This creates additional problems because of the number of pulses involved. Additionally, when the structures generate multiple pulses, they also create a new artificial clock in the design that must be analyzed by design tools.

Figure 40. Recommended Synchronous Pulse-Generation Technique



The pulse width is always equal to the clock period. This pulse generator is predictable, can be verified with timing analysis, and is easily moved to other architectures, devices, or speed grades.

5.2.2 Optimizing Clocking Schemes

Like combinational logic, clocking schemes have a large effect on the performance and reliability of a design.

Avoid using internally generated clocks (other than PLLs) wherever possible because they can cause functional and timing problems in the design. Clocks generated with combinational logic can introduce glitches that create functional problems, and the delay inherent in combinational logic can lead to timing problems.

Tip: Specify all clock relationships in the Quartus Prime software to allow for the best timing-driven optimizations during fitting and to allow correct timing analysis. Use clock setting assignments on any derived or internal clocks to specify their relationship to the base clock.

Use global device-wide, low-skew dedicated routing for all internally-generated clocks, instead of routing clocks on regular routing lines.

Avoid data transfers between different clocks wherever possible. If you require a data transfer between different clocks, use FIFO circuitry. You can use the clock uncertainty features in the Quartus Prime software to compensate for the variable delays between clock domains. Consider setting a clock setup uncertainty and clock hold uncertainty value of 10% to 15% of the clock delay.

The following sections provide specific examples and recommendations for avoiding clocking scheme problems.

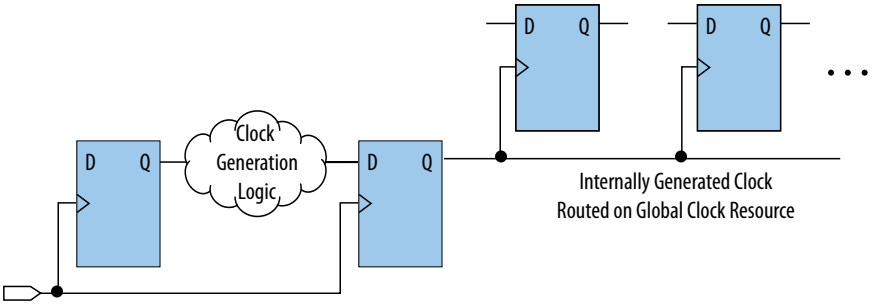
5.2.2.1 Register Combinational Logic Outputs

If you use the output from combinational logic as a clock signal or as an asynchronous reset signal, you can expect to see glitches in your design. In a synchronous design, glitches on data inputs of registers are normal events that have no consequences. However, a glitch or a spike on the clock input (or an asynchronous input) to a register can have significant consequences.

Narrow glitches can violate the register's minimum pulse width requirements. Setup and hold requirements might also be violated if the data input of the register changes when a glitch reaches the clock input. Even if the design does not violate timing requirements, the register output can change value unexpectedly and cause functional hazards elsewhere in the design.

To avoid these problems, you should always register the output of combinational logic before you use it as a clock signal.

Figure 41. Recommended Clock-Generation Technique



Registering the output of combinational logic ensures that glitches generated by the combinational logic are blocked at the data input of the register.

5.2.2.2 Avoid Asynchronous Clock Division

Designs often require clocks that you create by dividing a master clock. Most Intel FPGAs provide dedicated phase-locked loop (PLL) circuitry for clock division. Using dedicated PLL circuitry can help you avoid many of the problems that can be introduced by asynchronous clock division logic.

When you must use logic to divide a master clock, always use synchronous counters or state machines. Additionally, create your design so that registers always directly generate divided clock signals, and route the clock on global clock resources. To avoid glitches, do not decode the outputs of a counter or a state machine to generate clock signals. asynchronous

5.2.2.3 Avoid Ripple Counters

To simplify verification, avoid ripple counters in your design. In the past, FPGA designers implemented ripple counters to divide clocks by a power of two because the counters are easy to design and may use fewer gates than their synchronous counterparts.

Ripple counters use cascaded registers, in which the output pin of one register feeds the clock pin of the register in the next stage. This cascading can cause problems because the counter creates a ripple clock at each stage. These ripple clocks must be handled properly during timing analysis, which can be difficult and may require you to make complicated timing assignments in your synthesis and placement and routing tools.

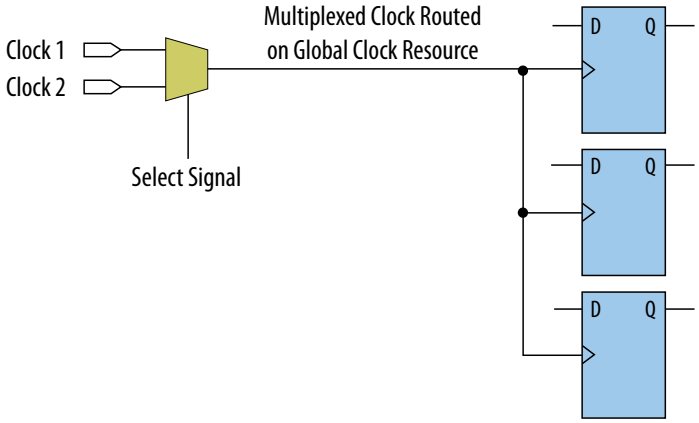
You can often use ripple clock structures to make ripple counters out of the smallest amount of logic possible. However, in all Intel devices supported by the Quartus Prime software, using a ripple clock structure to reduce the amount of logic used for a counter is unnecessary because the device allows you to construct a counter using one logic element per counter bit. You should avoid using ripple counters completely.

5.2.2.4 Use Multiplexed Clocks

Use clock multiplexing to operate the same logic function with different clock sources. In these designs, multiplexing selects a clock source.

For example, telecommunications applications that deal with multiple frequency standards often use multiplexed clocks.

Figure 42. Multiplexing Logic and Clock Sources



Adding multiplexing logic to the clock signal can create the problems addressed in the previous sections, but requirements for multiplexed clocks vary widely, depending on the application. Clock multiplexing is acceptable when the clock signal uses global clock routing resources and if the following criteria are met:

- The clock multiplexing logic does not change after initial configuration
- The design uses multiplexing logic to select a clock for testing purposes
- Registers are always reset when the clock switches
- A temporarily incorrect response following clock switching has no negative consequences

If the design switches clocks in real time with no reset signal, and your design cannot tolerate a temporarily incorrect response, you must use a synchronous design so that there are no timing violations on the registers, no glitches on clock signals, and no race conditions or other logical problems. By default, the Quartus Prime software optimizes and analyzes all possible paths through the multiplexer and between both internal clocks that may come from the multiplexer. This may lead to more restrictive analysis than required if the multiplexer is always selecting one particular clock. If you do not require the more complete analysis, you can assign the output of the multiplexer as a base clock in the Quartus Prime software, so that all register-to-register paths are analyzed using that clock.

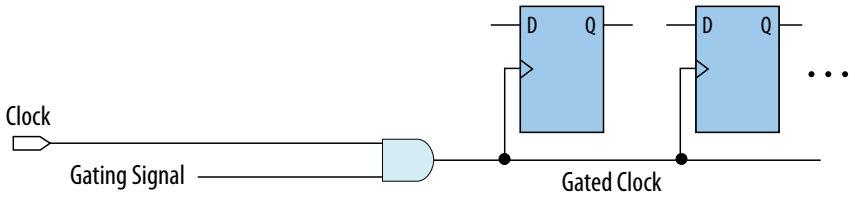
Tip: Use dedicated hardware to perform clock multiplexing when it is available, instead of using multiplexing logic. For example, you can use the clock-switchover feature or clock control block available in certain Intel FPGA devices. These dedicated hardware blocks ensure that you use global low-skew routing lines and avoid any possible hold time problems on the device due to logic delay on the clock line.

Note: For device-specific information about clocking structures, refer to the appropriate device data sheet or handbook on the Literature page of the Altera website.

5.2.2.5 Use Gated Clocks

Gated clocks turn a clock signal on and off using an enable signal that controls gating circuitry. When a clock is turned off, the corresponding clock domain is shut down and becomes functionally inactive.

Figure 43. Gated Clock



You can use gated clocks to reduce power consumption in some device architectures by effectively shutting down portions of a digital circuit when they are not in use. When a clock is gated, both the clock network and the registers driven by it stop toggling, thereby eliminating their contributions to power consumption. However, gated clocks are not part of a synchronous scheme and therefore can significantly increase the effort required for design implementation and verification. Gated clocks contribute to clock skew and make device migration difficult. These clocks are also sensitive to glitches, which can cause design failure.

Use dedicated hardware to perform clock gating rather than an AND or OR gate. For example, you can use the clock control block in newer Intel FPGA devices to shut down an entire clock network. Dedicated hardware blocks ensure that you use global routing with low skew, and avoid any possible hold time problems on the device due to logic delay on the clock line.

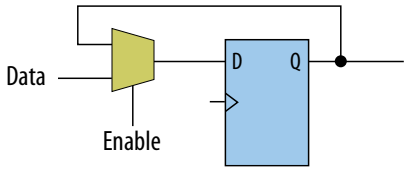
From a functional point of view, you can shut down a clock domain in a purely synchronous manner using a synchronous clock enable signal. However, when using a synchronous clock enable scheme, the clock network continues toggling. This practice does not reduce power consumption as much as gating the clock at the source does. In most cases, use a synchronous scheme.

5.2.2.6 Use Synchronous Clock Enables

To turn off a clock domain in a synchronous manner, use a synchronous clock enable signal. FPGAs efficiently support clock enable signals because there is a dedicated clock enable signal available on all device registers.

This scheme does not reduce power consumption as much as gating the clock at the source because the clock network keeps toggling, and performs the same function as a gated clock by disabling a set of registers. Insert a multiplexer in front of the data input of every register to either load new data, or copy the output of the register.

Figure 44. Synchronous Clock Enable

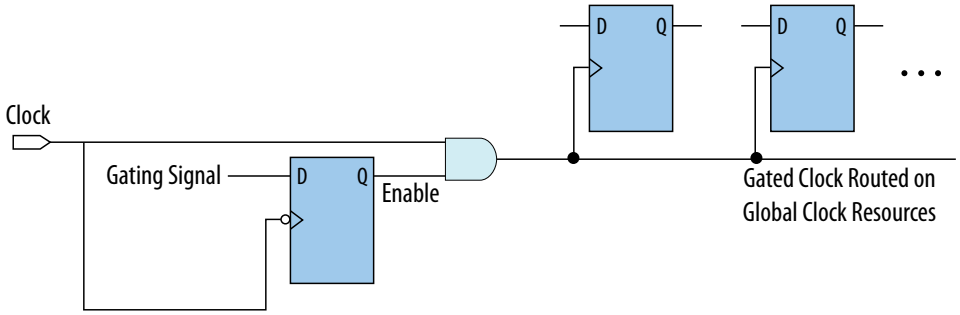


5.2.2.7 Recommended Clock-Gating Methods

Use gated clocks only when your target application requires power reduction and gated clocks provide the required reduction in your device architecture. If you must use clocks gated by logic, follow a robust clock-gating methodology and ensure the gated clock signal uses dedicated global clock routing.

You can gate a clock signal at the source of the clock network, at each register, or somewhere in between. Since the clock network contributes to switching power consumption, gate the clock at the source whenever possible to shut down the entire clock network instead of further along.

Figure 45. Recommended Clock-Gating Technique for Clock Active on Rising Edge



To generate a gated clock with the recommended technique, use a register triggered on the inactive edge of the clock. With this configuration, only one input of the gate changes at a time, preventing glitches or spikes on the output. If the clock is active on the rising edge, use an AND gate. Conversely, for a clock that is active on the falling edge, use an OR gate to gate the clock and register

Pay attention to the delay through the logic generating the enable signal, because the enable command must be ready in less than one-half the clock cycle. This might cause problems if the logic that generates the enable command is particularly complex, or if the duty cycle of the clock is severely unbalanced. However, careful management of the duty cycle and logic delay may be an acceptable solution when compared with problems created by other methods of gating clocks.

In the TimeQuest analyzer, ensure to apply a clock setting to the output of the AND gate. Otherwise, the timing analyzer might analyze the circuit using the clock path through the register as the longest clock path and the path that skips the register as the shortest clock path, resulting in artificial clock skew.

In certain cases, converting the gated clocks to clock enable pins may help reduce glitch and clock skew, and eventually produce a more accurate timing analysis. You can set the Quartus Prime software to automatically convert gated clocks to clock enable pins by turning on the **Auto Gated Clock Conversion** option. The conversion applies to two types of gated clocking schemes: single-gated clock and cascaded-gated clock.

Related Links

- Advanced Synthesis Settings on page 207
  The following section is a quick reference of all Advanced Synthesis Settings.
- Auto Gated Clock Conversion logic option
  In Quartus Prime Help

5.2.3 Optimizing Physical Implementation and Timing Closure

This section provides design and timing closure techniques for high speed or complex core logic designs with challenging timing requirements. These techniques may also be helpful for low or medium speed designs.

5.2.3.1 Planning Physical Implementation

When planning a design, consider the following elements of physical implementation:

- The number of unique clock domains and their relationships
- The amount of logic in each functional block
- The location and direction of data flow between blocks
- How data routes to the functional blocks between I/O interfaces

Interface-wide control or status signals may have competing or opposing constraints. For example, when a functional block's control or status signals interface with physical channels from both sides of the device. In such cases you must provide enough pipeline register stages to allow these signals to traverse the width of the device. In addition, you can structure the hierarchy of the design into separate logic modules for each side of the device. The side modules can generate and use registered control signals per side. This simplifies floorplanning, particularly in designs with transceivers, by placing per-side logic near the transceivers.

When adding register stages to pipeline control signals, turn off the **Auto Shift Register Replacement** option (**Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Synthesis)**) for these registers. By default, chains of registers can be converted to a RAM-based implementation based on performance and resource estimates. Since pipelining helps meet timing requirements over long distance, this assignment ensures that control signals are not converted.

5.2.3.2 Planning FPGA Resources

Your design requirements impact the use of FPGA resources. Plan functional blocks with appropriate global, regional, and dual-regional network signals in mind.

In general, after allocating the clocks in a design, use global networks for the highest fan-out control signals. When a global network signal distributes a high fan-out control signal, the global signal can drive logic anywhere in the device. Similarly, when using a regional network signal, the driven logic must be in one quadrant of the device, or half the device for a dual-regional network signal. Depending on data flow and physical locations of the data entry and exit between the I/Os and the device, restricting a functional block to a quadrant or half the device may not be practical for performance or resource requirements.

When floorplanning a design, consider the balance of different types of device resources, such as memory, logic, and DSP blocks in the main functional blocks. For example, if a design is memory intensive with a small amount of logic, it may be difficult to develop an effective floorplan. Logic that interfaces with the memory would have to spread across the chip to access the memory. In this case, it is important to use enough register stages in the data and control paths to allow signals to traverse the chip to access the physically disparate resources needed.

5.2.3.3 Optimizing Timing Closure

You can make changes to your design and constraints that help you achieve timing closure.

Whenever you change the project settings, you must balance any performance improvement of the setting against any potential increase in compilation time associated with the setting. You can view the performance gain versus runtime cost by reviewing the Fitter messages after design processing.

You can use physical synthesis optimizations for combinational logic, register retiming, and register duplication techniques to optimize your design for timing closure.

Click **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Fitter)** to turn on physical synthesis options.

- Physical synthesis for combinational logic—When the **Perform physical synthesis for combinational logic** is turned on, the report panel identifies logic that physical synthesis can modify. You can use this information to modify the design so that the associated optimization can be turned off to save compile time.
- Register duplication—This technique is most useful where registers have high fan-out, or where the fan-out is in physically distant areas of the device. Review the netlist optimizations report and consider manually duplicating registers automatically added by physical synthesis. You can also locate the original and duplicate registers in the Chip Planner. Compare their locations, and if the fan-out is improved, modify the code and turn off register duplication to save compile time.
- Register retiming—This technique is particularly useful where some combinatorial paths between registers exceed the timing goal while other paths fall short. If a design is already heavily pipelined, register retiming is less likely to provide significant performance gains since there should not be significantly unbalanced levels of logic across pipeline stages.

The application of appropriate timing constraints is essential to timing closure. Use the following general guidelines in applying timing constraints:

- Apply multicycle constraints in your design wherever single-cycle timing analysis is not required.
- Apply False Path constraints to all asynchronous clock domain crossings or resets in the design. This technique prevents overconstraining and the Fitter focuses only on critical paths to reduce compile time. However, overconstraining timing critical clock domains can sometimes provide better timing results and lower compile times than physical synthesis.
- Overconstrain rather than using physical synthesis when the slack improvement from physical synthesis is near zero. Overconstrain the frequency requirement on timing critical clock domains by using setup uncertainty.
- When evaluating the effect of constraint changes on performance and runtime, compile the design with at least three different seeds to determine the average performance and runtime effects. Different constraint combinations produce various results. Three samples or more establishes a performance trend. Modify your constraints based on performance improvement or decline.
- Leave settings at the default value whenever possible. Increasing performance constraints can increase the compile time significantly. While those increases may be necessary to close timing on a design, using the default settings whenever possible minimizes compile time.

5.2.3.4 Optimizing Critical Timing Paths

To close timing in high speed designs, review paths with the largest timing failures. Correcting a single, large timing failure can result in a very significant timing improvement.

Review the register placement and routing paths by clicking **Tools ➤ Chip Planner**. Large timing failures on high fan-out control signals can be caused by any of the following conditions:

- Sub-optimal use of global networks
- Signals that traverse the chip on local routing without pipelining
- Failure to correct high fan-out by register duplication

For high-speed and high-bandwidth designs, optimize speed by reducing bus width and wire usage. To reduce wire use, move the data as little as possible. For example, if a block of logic functions on a few bits of a word, store inactive bits in a FIFO or memory. Memory is cheaper and denser than registers and reduces wire usage.

5.2.4 Optimizing Power Consumption

The total FPGA power consumption is comprised of I/O power, core static power, and core dynamic power. Knowledge of the relationship between these components is fundamental in calculating the overall total power consumption.

You can use various optimization techniques and tools to minimize power consumption when applied during FPGA design implementation. The Quartus Prime software offers power-driven compilation features to fully optimize device power consumption. Power-driven compilation focuses on reducing your design's total power consumption using power-driven synthesis and power-driven placement and routing.

5.2.5 Managing Design Metastability

Metastability in PLD designs can be caused by the synchronization of asynchronous signals. You can use the Quartus Prime software to analyze the mean time between failures (MTBF) due to metastability, thus optimizing the design to improve the metastability MTBF. A high metastability MTBF indicates a more robust design.

5.3 Use Clock and Register-Control Architectural Features

In addition to following general design guidelines, you must code your design with the device architecture in mind. FPGAs provide device-wide clocks and register control signals that can improve performance.

5.3.1 Use Global Clock Network Resources

Intel FPGAs provide device-wide global clock routing resources and dedicated inputs. Use the FPGA's low-skew, high fan-out dedicated routing where available.

By assigning a clock input to one of these dedicated clock pins or with a Quartus Prime logic option to assign global routing, you can take advantage of the dedicated routing available for clock signals.

In an ASIC design, you should balance the clock delay as it is distributed across the device. Because Intel FPGAs provide device-wide global clock routing resources and dedicated inputs, there is no need to manually balance delays on the clock network.

You should limit the number of clocks in your design to the number of dedicated global clock resources available in your FPGA. Clocks feeding multiple locations that do not use global routing may exhibit clock skew across the device that could lead to timing problems. In addition, when you use combinational logic to generate an internal clock, it adds delays on the clock path. In some cases, delay on a clock line can result in a clock skew greater than the datapath length between two registers. If the clock skew is greater than the data delay, you violate the timing parameters of the register (such as hold time requirements) and the design does not function correctly.

FPGAs offer a number of low-skew global routing resources to distribute high fan-out signals to help with the implementation of large designs with many clock domains. Many large FPGA devices provide dedicated global clock networks, regional clock networks, and dedicated fast regional clock networks. These clocks are organized into a hierarchical clock structure that allows many clocks in each device region with low skew and delay. There are typically several dedicated clock pins to drive either global or regional clock networks, and both PLL outputs and internal clocks can drive various clock networks.

To reduce clock skew in a given clock domain and ensure that hold times are met in that clock domain, assign each clock signal to one of the global high fan-out, low-skew clock networks in the FPGA device. The Quartus Prime software automatically uses global routing for high fan-out control signals, PLL outputs, and signals feeding the global clock pins on the device. You can make explicit Global Signal logic option settings by turning on the **Global Signal** option setting. Use this option when it is necessary to force the software to use the global routing for particular signals.

To take full advantage of these routing resources, the sources of clock signals in a design (input clock pins or internally-generated clocks) need to drive only the clock input ports of registers. In older Intel device families, if a clock signal feeds the data ports of a register, the signal may not be able to use dedicated routing, which can lead

to decreased performance and clock skew problems. In general, allowing clock signals to drive the data ports of registers is not considered synchronous design and can complicate timing analysis.

5.3.2 Use Global Reset Resources

ASIC designs may use local resets to avoid long routing delays. Take advantage of the device-wide asynchronous reset pin available on most FPGAs to eliminate these problems. This reset signal provides low-skew routing across the device.

The following are three types of resets used in synchronous circuits:

- Synchronous Reset
- Asynchronous Reset
- Synchronized Asynchronous Reset—preferred when designing an FPGA circuit

5.3.2.1 Use Synchronous Resets

The synchronous reset ensures that the circuit is fully synchronous. You can easily time the circuit with the Quartus Prime TimeQuest analyzer.

Because clocks that are synchronous to each other launch and latch the reset signal, the data arrival and data required times are easily determined for proper slack analysis. The synchronous reset is easier to use with cycle-based simulators.

There are two methods by which a reset signal can reach a register; either by being gated in with the data input, or by using an LAB-wide control signal (`sync lr`). If you use the first method, you risk adding an additional gate delay to the circuit to accommodate the reset signal, which causes increased data arrival times and negatively impacts setup slack. The second method relies on dedicated routing in the LAB to each register, but this is slower than an asynchronous reset to the same register.

Figure 46. Synchronous Reset

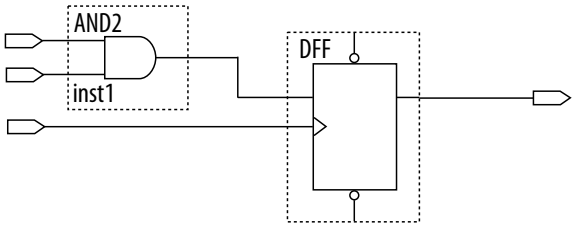
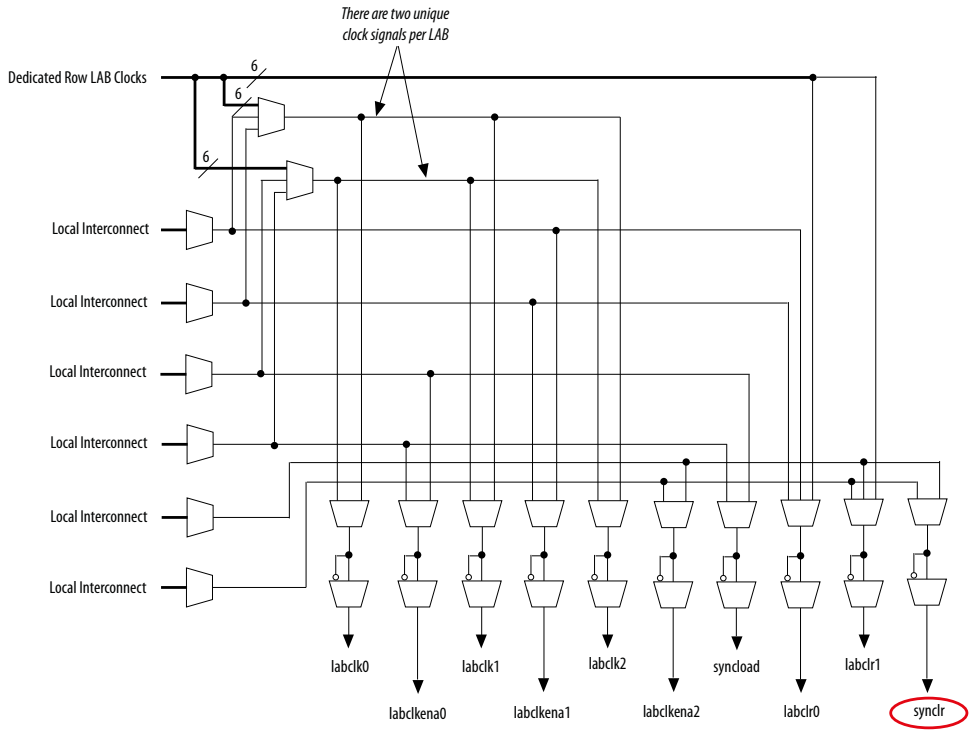
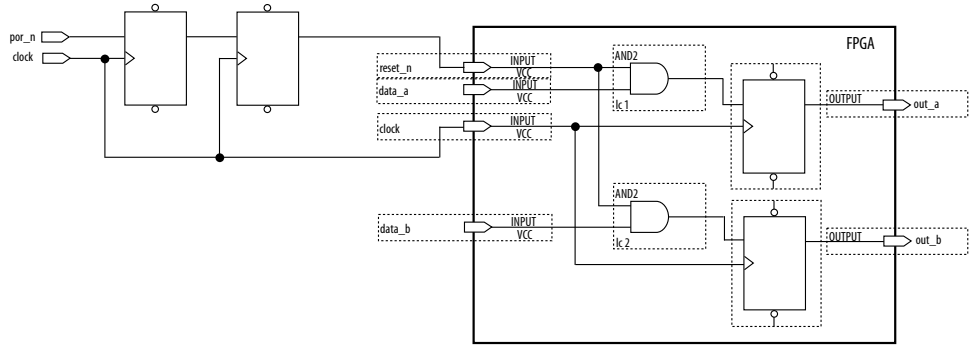


Figure 47. LAB-Wide Control Signals



Consider two types of synchronous resets when you examine the timing analysis of synchronous resets—externally synchronized resets and internally synchronized resets. Externally synchronized resets are synchronized to the clock domain outside the FPGA, and are not very common. A power-on asynchronous reset is dual-rank synchronized externally to the system clock and then brought into the FPGA. Inside the FPGA, gate this reset with the data input to the registers to implement a synchronous reset.

Figure 48. Externally Synchronized Reset



The following example shows the Verilog HDL equivalent of the schematic. When you use synchronous resets, the reset signal is not put in the sensitivity list.

The following example shows the necessary modifications that you should make to the internally synchronized reset.

Example 56. Verilog HDL Code for Externally Synchronized Reset

```
module sync_reset_ext (
         input   clock,
         input   reset_n,
         input   data_a,
         input   data_b,
         output  out_a,
         output  out_b
         );
reg      reg1, reg2
assign   out_a  = reg1;
assign   out_b  = reg2;
always @ (posedge clock)
begin
     if (!reset_n)
     begin
         reg1     <= 1'b0;
         reg2     <= 1'b0;
    end
    else
    begin
         reg1     <= data_a;
         reg2     <= data_b;
    end
end
endmodule     //  sync_reset_ext
```

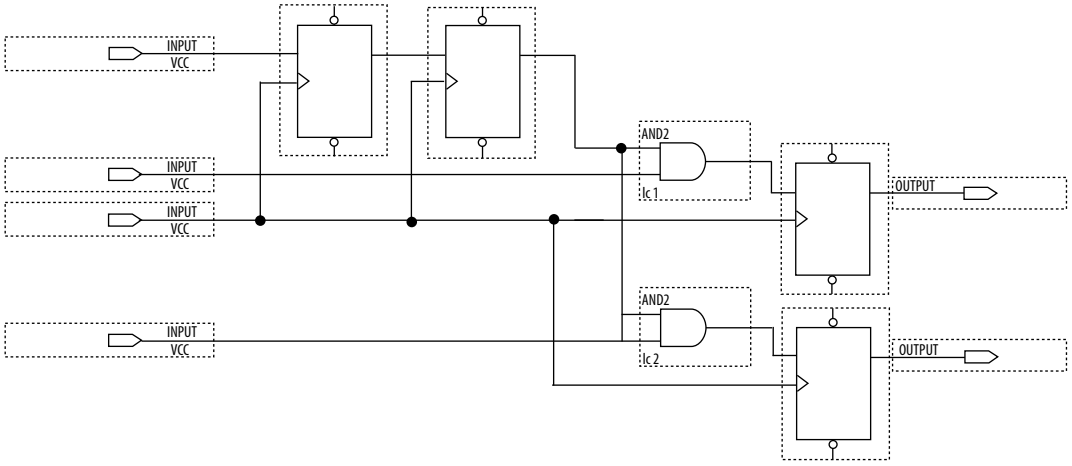
The following example shows the constraints for the externally synchronous reset. Because the external reset is synchronous, you only need to constrain the `reset_n` signal as a normal input signal with `set_input_delay` constraint for `-max` and `-min`.

Example 57. SDC Constraints for Externally Synchronized Reset

```
# Input clock - 100 MHz
create_clock [get_ports {clock}] \
        -name {clock} \
        -period 10.0 \
        -waveform {0.0 5.0}
# Input constraints on low-active reset
# and data
set_input_delay 7.0 \
        -max \
        -clock [get_clocks {clock}] \
        [get_ports {reset_n data_a data_b}]
set_input_delay 1.0 \
        -min \
        -clock [get_clocks {clock}] \
        [get_ports {reset_n data_a data_b}]
```

More often, resets coming into the device are asynchronous, and must be synchronized internally before being sent to the registers.

Figure 49. Internally Synchronized Reset



The following example shows the Verilog HDL equivalent of the schematic. Only the clock edge is in the sensitivity list for a synchronous reset.

Example 58. Verilog HDL Code for Internally Synchronized Reset

```
module sync_reset (
        input clock,
        input reset_n,
        input data_a,
        input data_b,
        output out_a,
        output out_b
        );
reg     reg1, reg2
reg     reg3, reg4

assign     out_a = reg1;
assign     out_b = reg2;
assign     rst_n = reg4;

always @ (posedge clock)
begin
    if (!rst_n)
    begin
        reg1 <= 1'bo;
        reg2 <= 1'b0;
    end
    else
    begin
        reg1 <= data_a;
        reg2 <= data_b;
    end
end

always @ (posedge clock)
begin
    reg3 <= reset_n;
    reg4 <= reg3;
end
endmodule // sync_reset
```

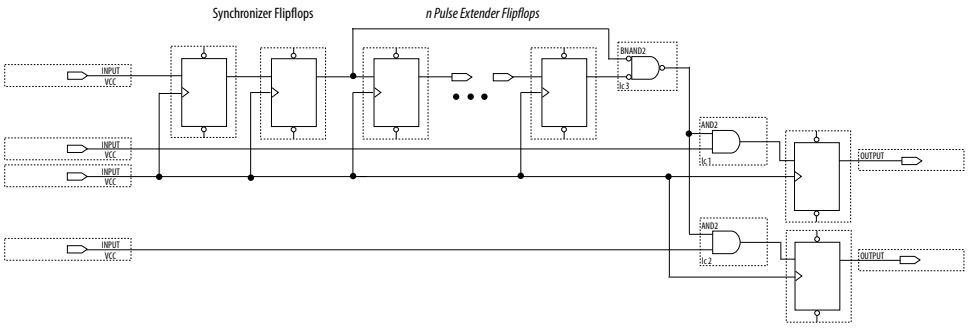
The SDC constraints are similar to the external synchronous reset, except that the input reset cannot be constrained because it is asynchronous. Cut the input path with a `set_false_path` statement to avoid these being considered as unconstrained paths.

Example 59. SDC Constraints for Internally Synchronized Reset

```
# Input clock - 100 MHz
create_clock [get_ports {clock}] \
        -name {clock} \
        -period 10.0 \
        -waveform {0.0 5.0}
# Input constraints on data
set_input_delay 7.0 \
        -max \
        -clock [get_clocks {clock}] \
        [get_ports {data_a data_b}]
set_input_delay 1.0 \
        -min \
        -clock [get_clocks {clock}] \
        [get_ports {data_a data_b}]
# Cut the asynchronous reset input
set_false_path \
        -from [get_ports {reset_n}] \
        -to [all_registers]
```

An issue with synchronous resets is their behavior with respect to short pulses (less than a period) on the asynchronous input to the synchronizer flipflops. This can be a disadvantage because the asynchronous reset requires a pulse width of at least one period wide to guarantee that it is captured by the first flipflop. However, this can also be viewed as an advantage in that this circuit increases noise immunity. Spurious pulses on the asynchronous input have a lower chance of being captured by the first flipflop, so the pulses do not trigger a synchronous reset. In some cases, you might want to increase the noise immunity further and reject any asynchronous input reset that is less than n periods wide to debounce an asynchronous input reset.

Figure 50. Internally Synchronized Reset with Pulse Extender



Junction dots indicate the number of stages. You can have more flipflops to get a wider pulse that spans more clock cycles.

Many designs have more than one clock signal. In these cases, use a separate reset synchronization circuit for each clock domain in the design. When you create synchronizers for PLL output clocks, these clock domains are not reset until you lock the PLL and the PLL output clocks are stable. If you use the reset to the PLL, this reset does not have to be synchronous with the input clock of the PLL. You can use an asynchronous reset for this. Using a reset to the PLL further delays the assertion of a synchronous reset to the PLL output clock domains when using internally synchronized resets.

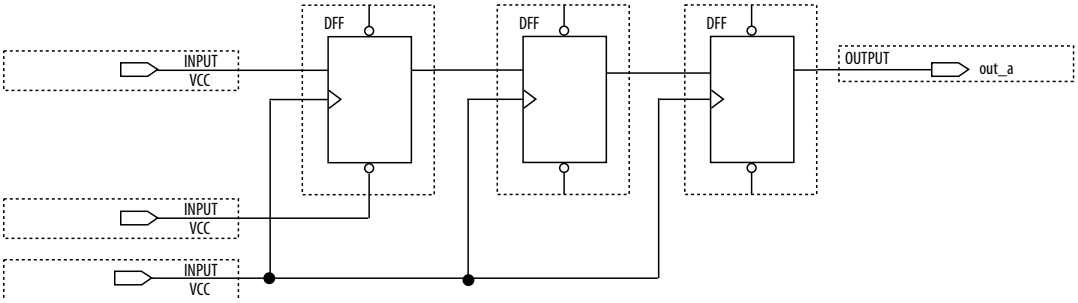
5.3.2.2 Using Asynchronous Resets

Asynchronous resets are the most common form of reset in circuit designs, as well as the easiest to implement. Typically, you can insert the asynchronous reset into the device, turn on the global buffer, and connect to the asynchronous reset pin of every register in the device.

This method is only advantageous under certain circumstances—you do not need to always reset the register. Unlike the synchronous reset, the asynchronous reset is not inserted in the datapath, and does not negatively impact the data arrival times between registers. Reset takes effect immediately, and as soon as the registers receive the reset pulse, the registers are reset. The asynchronous reset is not dependent on the clock.

However, when the reset is deasserted and does not pass the recovery (μt_{SU}) or removal (μt_H) time check (the TimeQuest analyzer recovery and removal analysis checks both times), the edge is said to have fallen into the metastability zone. Additional time is required to determine the correct state, and the delay can cause the setup time to fail to register downstream, leading to system failure. To avoid this, add a few follower registers after the register with the asynchronous reset and use the output of these registers in the design. Use the follower registers to synchronize the data to the clock to remove the metastability issues. You should place these registers close to each other in the device to keep the routing delays to a minimum, which decreases data arrival times and increases MTBF. Ensure that these follower registers themselves are not reset, but are initialized over a period of several clock cycles by "flushing out" their current or initial state.

Figure 51. Asynchronous Reset with Follower Registers



The following example shows the equivalent Verilog HDL code. The active edge of the reset is now in the sensitivity list for the procedural block, which infers a clock enable on the follower registers with the inverse of the reset signal tied to the clock enable. The follower registers should be in a separate procedural block as shown using non-blocking assignments.

Example 60. Verilog HDL Code of Asynchronous Reset with Follower Registers

```
module async_reset (
        input   clock,
        input   reset_n,
        input   data_a,
        output   out_a,
        );
reg     reg1, reg2, reg3;
assign  out_a  = reg3;
always @ (posedge clock, negedge reset_n)
begin
    if (!reset_n)
        reg1    <= 1'b0;
    else
        reg1    <= data_a;
end
always @ (posedge clock)
begin
    reg2    <= reg1;
    reg3    <= reg2;
end
endmodule  //  async_reset
```

You can easily constrain an asynchronous reset. By definition, asynchronous resets have a non-deterministic relationship to the clock domains of the registers they are resetting. Therefore, static timing analysis of these resets is not possible and you can use the `set_false_path` command to exclude the path from timing analysis. Because the relationship of the reset to the clock at the register is not known, you cannot run recovery and removal analysis in the TimeQuest analyzer for this path. Attempting to do so even without the false path statement results in no paths reported for recovery and removal.

Example 61. SDC Constraints for Asynchronous Reset

```
# Input clock - 100 MHz
create_clock [get_ports {clock}] \
        -name {clock} \
        -period 10.0 \
        -waveform {0.0 5.0}
```

```
# Input constraints on data
set_input_delay 7.0 \
        -max \
        -clock [get_clocks {clock}]\
        [get_ports {data_a}]
set_input_delay 1.0 \
        -min \
        -clock [get_clocks {clock}] \
        [get_ports {data_a}]
# Cut the asynchronous reset input
set_false_path \
        -from [get_ports {reset_n}] \
        -to [all_registers]
```

The asynchronous reset is susceptible to noise, and a noisy asynchronous reset can cause a spurious reset. You must ensure that the asynchronous reset is debounced and filtered. You can easily enter into a reset asynchronously, but releasing a reset asynchronously can lead to potential problems (also referred to as "reset removal") with metastability, including the hazards of unwanted situations with synchronous circuits involving feedback.

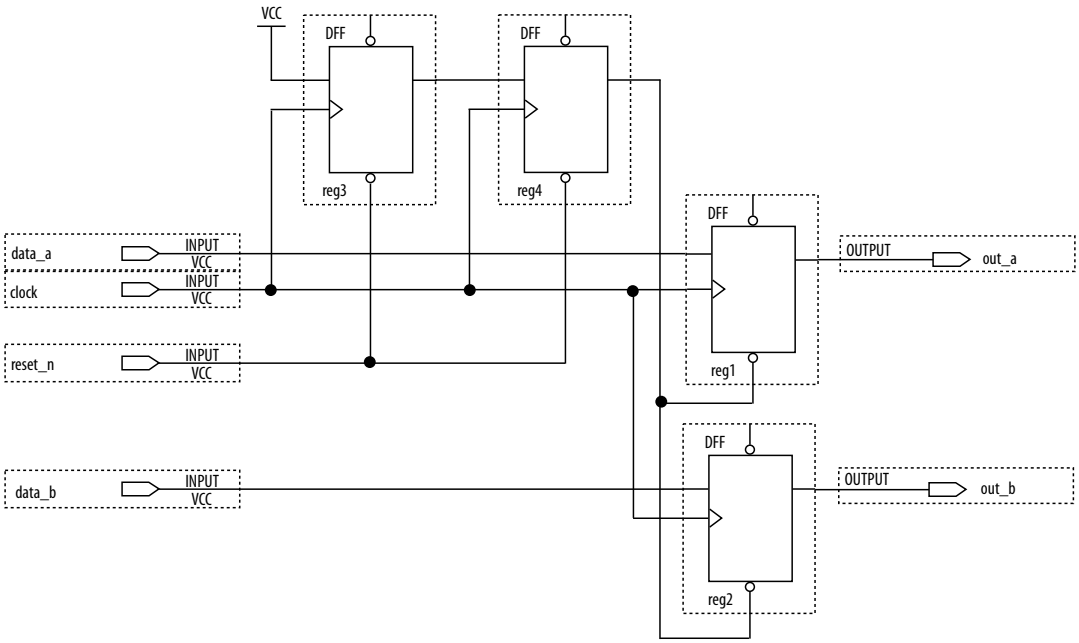
5.3.2.3 Use Synchronized Asynchronous Reset

To avoid potential problems associated with purely synchronous resets and purely asynchronous resets, you can use synchronized asynchronous resets. Synchronized asynchronous resets combine the advantages of synchronous and asynchronous resets.

These resets are asynchronously asserted and synchronously deasserted. This takes effect almost instantaneously, and ensures that no datapath for speed is involved. Also, the circuit is synchronous for timing analysis and is resistant to noise.

The following example shows a method for implementing the synchronized asynchronous reset. You should use synchronizer registers in a similar manner as synchronous resets. However, the asynchronous reset input is gated directly to the CLRN pin of the synchronizer registers and immediately asserts the resulting reset. When the reset is deasserted, logic "1" is clocked through the synchronizers to synchronously deassert the resulting reset.

Figure 52. Schematic of Synchronized Asynchronous Reset



The following example shows the equivalent Verilog HDL code. Use the active edge of the reset in the sensitivity list for the blocks.

Example 62. Verilog HDL Code for Synchronized Asynchronous Reset

```
module sync_async_reset (
        input    clock,
        input    reset_n,
        input    data_a,
        input    data_b,
        output   out_a,
        output   out_b
        );
reg     reg1, reg2;
reg     reg3, reg4;
assign  out_a    = reg1;
assign  out_b    = reg2;
assign  rst_n    = reg4;
always @ (posedge clock, negedge reset_n)
begin
    if (!reset_n)
    begin
       reg3     <= 1'b0;
       reg4     <= 1'b0;
    end
    else
    begin
       reg3     <= 1'b1;
       reg4     <= reg3;
    end
end
always @ (posedge clock, negedge rst_n)
begin
    if (!rst_n)
    begin
       reg1     <= 1'b0;
       reg2     <= 1;b0;
```

```
    end
    else
    begin
       reg1     <= data_a;
       reg2     <= data_b;
    end
end
endmodule  // sync_async_reset
```

To minimize the metastability effect between the two synchronization registers, and to increase the MTBF, the registers should be located as close as possible in the device to minimize routing delay. If possible, locate the registers in the same logic array block (LAB). The input reset signal (`reset_n`) must be excluded with a `set_false_path` command:

```
set_false_path -from [get_ports {reset_n}] -to [all_registers]
```

The `set_false_path` command used with the specified constraint excludes unnecessary input timing reports that would otherwise result from specifying an input delay on the reset pin.

The instantaneous assertion of synchronized asynchronous resets is susceptible to noise and runt pulses. If possible, you should debounce the asynchronous reset and filter the reset before it enters the device. The circuit ensures that the synchronized asynchronous reset is at least one full clock period in length. To extend this time to *n* clock periods, you must increase the number of synchronizer registers to *n* + 1. You must connect the asynchronous input reset (`reset_n`) to the CLRN pin of all the synchronizer registers to maintain the asynchronous assertion of the synchronized asynchronous reset.

5.3.3 Avoid Asynchronous Register Control Signals

Avoid using an asynchronous load signal if the design target device architecture does not include registers with dedicated circuitry for asynchronous loads. Also, avoid using both asynchronous clear and preset if the architecture provides only one of these control signals.

Some Intel devices directly support an asynchronous clear function, but not a preset or load function. When the target device does not directly support the signals, the synthesis or placement and routing software must use combinational logic to implement the same functionality. In addition, if you use signals in a priority other than the inherent priority in the device architecture, combinational logic may be required to implement the necessary control signals. Combinational logic is less efficient and can cause glitches and other problems; it is best to avoid these implementations.

5.4 Implementing Embedded RAM

Intel's dedicated memory architecture offers many advanced features that you can enable with Intel-provided IP cores. Use synchronous memory blocks for your design, so that the blocks can be mapped directly into the device dedicated memory blocks.

You can use single-port, dual-port, or three-port RAM with a single- or dual-clocking method. You should not infer the asynchronous memory logic as a memory block or place the asynchronous memory logic in the dedicated memory block, but implement the asynchronous memory logic in regular logic cells.

Intel memory blocks have different read-during-write behaviors, depending on the targeted device family, memory mode, and block type. Read-during-write behavior refers to read and write from the same memory address in the same clock cycle; for example, you read from the same address to which you write in the same clock cycle.

You should check how you specify the memory in your HDL code when you use read-during-write behavior. The HDL code that describes the read returns either the old data stored at the memory location, or the new data being written to the memory location.

In some cases, when the device architecture cannot implement the memory behavior described in your HDL code, the memory block is not mapped to the dedicated RAM blocks, or the memory block is implemented using extra logic in addition to the dedicated RAM block. Implement the read-during-write behavior using single-port RAM in Arria GX devices and the Cyclone and Stratix series of devices to avoid this extra logic implementation.

In many synthesis tools, you can specify that the read-during-write behavior is not important to your design; if, for example, you never read and write from the same address in the same clock cycle.

Related Links

Inferring RAM functions from HDL Code on page 107
To infer RAM functions, synthesis tools recognize certain types of HDL code and map the detected code to technology-specific implementations.

5.5 Document Revision History

Table 29. Document Revision History

Date	Version	Changes
2017.05.08	17.0.0	• Removed information about Integrated Synthesis. • Removed information about quartus_drc.
2016.10.31	16.1.0	• Implemented Intel rebranding.
2016.05.03	16.0.0	• Replaced Internally Synchronized Reset code sample with corrected version. • Removed information about deprecated physical synthesis options. • Removed information about unsupported Design Assistant.
2015.11.02	15.1.0	• Changed instances of *Quartus II* to *Quartus Prime*.
2014.12.15	14.1.0	Updated location of Fitter Settings, Analysis & Synthesis Settings, and Physical Optimization Settings to Compiler Settings.
June 2014	14.0.0	Removed references to obsolete MegaWizard Plug-In Manager.
November 2013	13.1.0	Removed HardCopy device information.
May 2013	13.0.0	Removed PrimeTime support.
June 2012	12.0.0	Removed survey link.
November 2011	11.0.1	Template update.

continued...

Date	Version	Changes
May 2011	11.0.0	Added information to Reset Resources .
December 2010	10.1.0	• Title changed from Design Recommendations for Intel Devices and the Quartus Prime Design Assistant. • Updated to new template. • Added references to Quartus Prime Help for "Metastability" on page 9–13 and "Incremental Compilation" on page 9–13. • Removed duplicated content and added references to Quartus Prime Help for "Custom Rules" on page 9–15.
July 2010	10.0.0	• Removed duplicated content and added references to Quartus Prime Help for Design Assistant settings, Design Assistant rules, Enabling and Disabling Design Assistant Rules, and Viewing Design Assistant reports. • Removed information from "Combinational Logic Structures" on page 5–4 • Changed heading from "Design Techniques to Save Power" to "Power Optimization" on page 5–12 • Added new "Metastability" section • Added new "Incremental Compilation" section • Added information to "Reset Resources" on page 5–23 • Removed "Referenced Documents" section
November 2009	9.1.0	• Removed documentation of obsolete rules.
March 2009	9.0.0	• No change to content.
November 2008	8.1.0	• Changed to 8-1/2 x 11 page size • Added new section "Custom Rules Coding Examples" on page 5–18 • Added paragraph to "Recommended Clock-Gating Methods" on page 5–11 • Added new section: "Design Techniques to Save Power" on page 5–12
May 2008	8.0.0	• Updated Figure 5–9 on page 5–13; added custom rules file to the flow • Added notes to Figure 5–9 on page 5–13 • Added new section: "Custom Rules Report" on page 5–34 • Added new section: "Custom Rules" on page 5–34 • Added new section: "Targeting Embedded RAM Architectural Features" on page 5–38 • Minor editorial updates throughout the chapter • Added hyperlinks to referenced documents throughout the chapter

Related Links

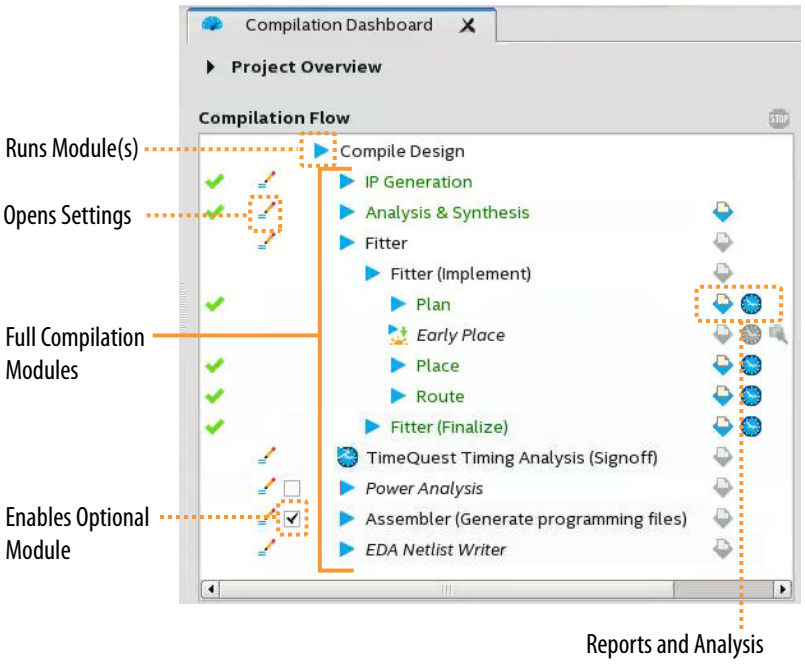
Altera Documentation Archive

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

6 Design Compilation

The Intel Quartus Prime Compiler's integrated modules synthesize, place, and route your design before ultimately generating a device programming file. The Compiler supports a wide variety of high-level, RTL, and schematic design entry methods. The Compiler includes the IP Generation, Analysis & Synthesis, Fitter, Timing Analyzer, and Assembler modules. Use the Compilation Dashboard for quick access to all Compiler controls, settings, and reports.

Figure 53. Compilation Dashboard



The Quartus Prime Pro Edition Compiler supports these unique features:

- Latest compilation support for Intel Arria 10, and Cyclone 10 GX devices.
- Incremental Fitter optimization—optimize after each Fitter stage to maximize performance and shorten total compilation time.
- Enhanced synthesis Engine—`quartus_syn` design synthesis implements stricter language parsing, new RAM inference, enhanced algorithms, and true parallel synthesis.
- Device Partial Reconfiguration—reconfigures a portion of the FPGA dynamically, while the remaining FPGA continues to function.

6.1 Compilation Overview

The Compiler is modular, allowing you to run only the process that you need. Each Compiler module performs a specific function in the full compilation process. When you run any module, the Compiler runs any prerequisite modules automatically.

Table 30. Compilation Modules

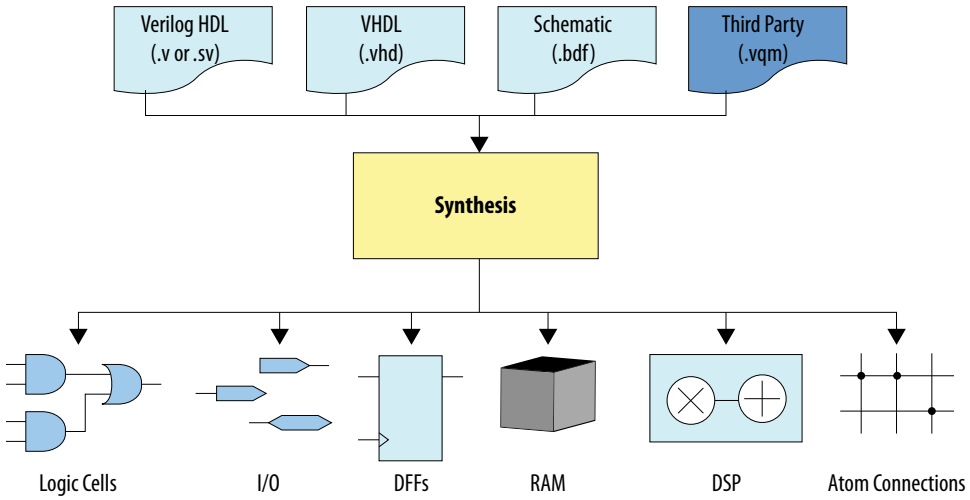
Compilation Process	Description
IP Generation	Identifies the status and version IP components in the project.
Analysis & Synthesis	Synthesizes, optimizes, minimizes, and maps design logic to device resources. Analysis & Elaboration is a stage of Analysis & Synthesis. This stage checks for design file and project errors.
Fitter (Place & Route)	Assigns the placement and routing of the design to specific device resources, while honoring timing and placement constraints. The Fitter includes the following stages: • Plan—performs periphery placement and routing. • Early Place—begins core placement. • Place—performs full logic placement. • Route—fully routes the design. • Finalize—converts unnecessary tiles to High-Speed or Low-Power.
TimeQuest Timing Analyzer	Analyzes and validates the timing performance of all design logic.
Power Analysis	Optional module that estimates device power consumption. Specify the electrical standard on each I/O cell and the board trace model on each I/O standard in your design.
Assembler	Converts the Fitter's placement and routing assignments into a programming image for the FPGA device.
EDA Netlist Writer	The EDA Netlist Writer generates output files for use in other EDA tools during full compilation flow.

6.1.1 Design Synthesis

Design synthesis is the process that translates design source files into an atom netlist for mapping to device resources. The Quartus Prime Compiler synthesizes standards-compliant Verilog HDL (`.v`), VHDL (`.vhd`), and SystemVerilog (`.sv`). The Compiler also synthesizes Block Design File (`.bdf`) schematic files, and the Verilog Quartus Mapping (`.vqm`) files generated by other EDA tools.

Synthesis examines the logical completeness and consistency of the design, and checks for boundary connectivity and syntax errors. Synthesis also minimizes and optimizes design logic. For example, synthesis infers D flip flops, latches, and state machines from "behavioral" languages, such as Verilog HDL, VHDL, and SystemVerilog. Synthesis may replace operators, such as + or –, with modules from the Quartus Prime IP Library, when advantageous. During synthesis, the Compiler may change or remove user logic and design nodes. Quartus Prime synthesis minimizes gate count, removes redundant logic, and ensures efficient use of device resources.

Figure 54. Design Synthesis



At the end of synthesis the Compiler generates an atom netlist. Atom refers to the most basic hardware resource in the FPGA device. Atoms include logic cells organized into look-up tables, D flip flops, I/O pins, block memory resources, DSP blocks, and the connections required to connect the atoms. The atom netlist is a database of the atom elements that design synthesis requires to implement the design in silicon.

The Analysis & Synthesis module of the Compiler synthesizes design files and creates one or more project databases for each design partition. You can specify various settings that affect synthesis processing.

6.1.2 Design Place and Route

The Compiler's Fitter module (`quartus_fit`) performs design placement and routing. During place and route, the Fitter determines the best placement and routing of logic in the target FPGA device, while respecting any Fitter settings or constraints that you specify.

By default, the Fitter selects appropriate resources, interconnection paths and pin locations. If you assign logic to specific device resources, the Fitter attempts to match those requirements, and then fits and optimizes any remaining unconstrained design logic. If the Fitter cannot fit the design in the current target device, the Fitter terminates compilation and issues an error message.

The Quartus Prime Pro Edition Fitter introduces a hybrid placement technique that combines analytical and annealing placement techniques. Analytical placement determines an initial mathematical starting placement. The annealing technique then fine-tunes logic block placement in high resource utilization scenarios.

The Quartus Prime Pro Edition Compiler allows control and optimization of each individual Fitter stage, including the Plan, Early Place, Place, and Route stages. The Compiler generates a snapshot and detailed reports for each stage. After running a Fitter stage, view detailed report data and analyze the timing of that stage.

Related Links

- Running the Fitter on page 193

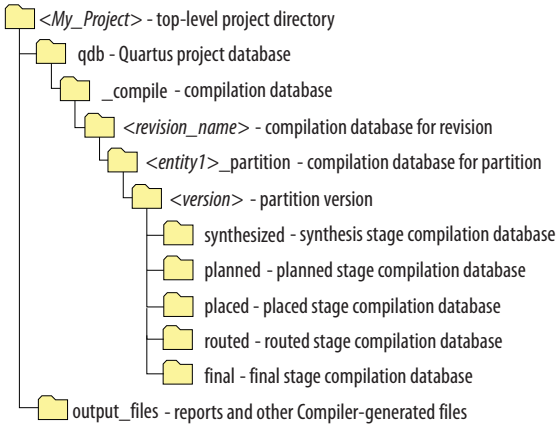
- Viewing Fitter Reports on page 196

6.1.3 Compilation Hierarchy

The Quartus Prime Pro Edition Compiler generates a new hierarchical project structure that isolates the compilation results of each design entity within a design partition. This hierarchical structure allows you to optimize specific design elements without impacting placement and routing in other partitions.

The Compiler fully preserves routing and placement within a partition. Changes to other portions of the design hierarchy do not impact the partition. The hierarchical project structure also supports distributed work groups and compilation processing across multiple machines.

Figure 55. Project Directory Hierarchy



6.1.4 Programming File Generation

The Compiler's Assembler module generates files for device programming. Run the Assembler automatically as part of a full compilation, or run the Assembler module independently after design place and route. After running the Assembler, use the Programmer to download configuration data to a device. The Assembler generates one or more of the following files according to your specification in the **Device & Pin Options** dialog box.

Table 31. Assembler Generated Programming Files

Programming File	Description
SRAM Object Files (`.sof`)	A binary file containing the data for configuring all SRAM-based Intel FPGA devices, including Arria 10 devices.
Programmer Object Files (`.pof`)	A binary file containing the data for programming an EEPROM-based Intel configuration device. For example, the EPCS16 and EPCS64 devices, which configure SRAM-based Intel FPGA devices.
Hexadecimal (Intel-Format) Output Files (`.hexout`)	Contains configuration data that you can program into a parallel data source, such as an EPROM or a mass storage device, which configures an SRAM-based Intel FPGA device.

continued...

Programming File	Description
Tabular Text Files (.ttf)	Contains configuration data that an intelligent external controller uses to configure an SRAM-based Intel FPGA device.
Raw Binary Files (.rbf)	
Serial Vector Format File (.svf)	

Related Links

Generating Programming Files on page 199

6.1.5 Reducing Compilation Time

The Quartus Prime Pro Edition software supports various strategies to reduce overall design compilation time. Running a full compilation including all Compiler modules on a large design can be time consuming. Use any the following techniques to reduce the overall compilation times of your design:

- Rapid Recompilation of changed blocks—the Compiler reuses previous compilation results and does not reprocess unchanged design blocks.
- Parallel compilation—the Compiler detects and uses multiple processors to reduce compilation time (for systems with multiple processor cores).
- Incremental optimization—breaks compilation into separate stages, allowing iterative analysis of results and optimization of settings at various compilation stages, prior to running a full compilation.

6.2 Compilation Flows

The Quartus Prime Pro Edition Compiler supports a variety of flows to help you maximize performance and minimize compilation processing time. The modular Compiler is flexible and efficient, allowing you to run all modules in sequence with a single command, or to run and optimize each stage of compilation separately.

As you develop and optimize your design, run only the Compiler stages that you need, rather than waiting for full compilation. Run full compilation only when your design is complete and you are ready to run all Compiler modules and generate a device programming image.

Table 32. Compilation Flows

Compiler Flow	Function
Early Place Flow	Begins core logic placement. Run Early Place to review initial high-level placement of design elements in the Chip Planner. This information is useful to guide your floorplanning decisions.
Implement Flow	Runs the Plan, Early Place, Place, and Route stages. Run this flow when you are ready to implement placement, routing, and retiming.
Finalize Flow	Runs the Plan, Early Place, Place, and Route Compilation stages. Run this flow when you are ready to verify final timing closure results and generate a device programming file to implement the design in the target device.
Incremental Optimization Flow	Incremental optimization allows you to stop processing after each stage, analyze the results, and adjust settings or RTL before proceeding to the next compilation stage. This iterative flow optimizes at each stage, without waiting for full compilation results.

continued...

Compiler Flow	Function
Full Compilation Flow	Launches all Compiler modules in sequence to synthesize, fit, analyze final timing, and generate a device programming file.
Partial Reconfiguration	Reconfigures a portion of the FPGA dynamically, while the remaining FPGA design continues to function.
Block-Level Design Flows	Supports preservation and reuse of design blocks in one or more projects. You can reuse synthesized, placed, or routed design blocks within the same project, or export the block to other projects. Reusable design blocks can include device core or periphery resources.

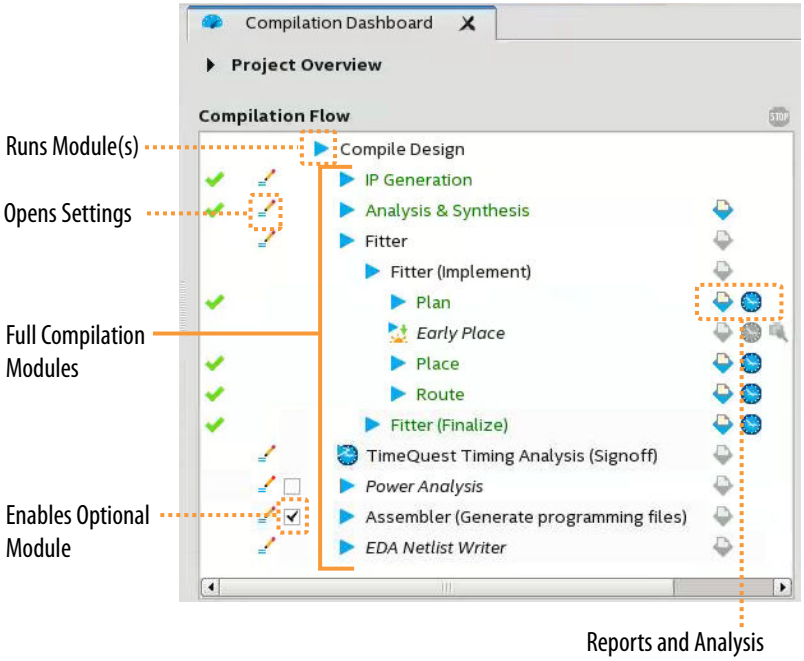
Related Links

- Incremental Optimization Flow on page 188
- Creating a Partial Reconfiguration Design
- Block-Based Design Flows

6.2.1 Full Compilation Flow

Full compilation uses a single command to launch all Compiler modules in sequence, running design synthesis, fitting, timing analysis, and programming file generation.

Figure 56. Full Compilation in Dashboard

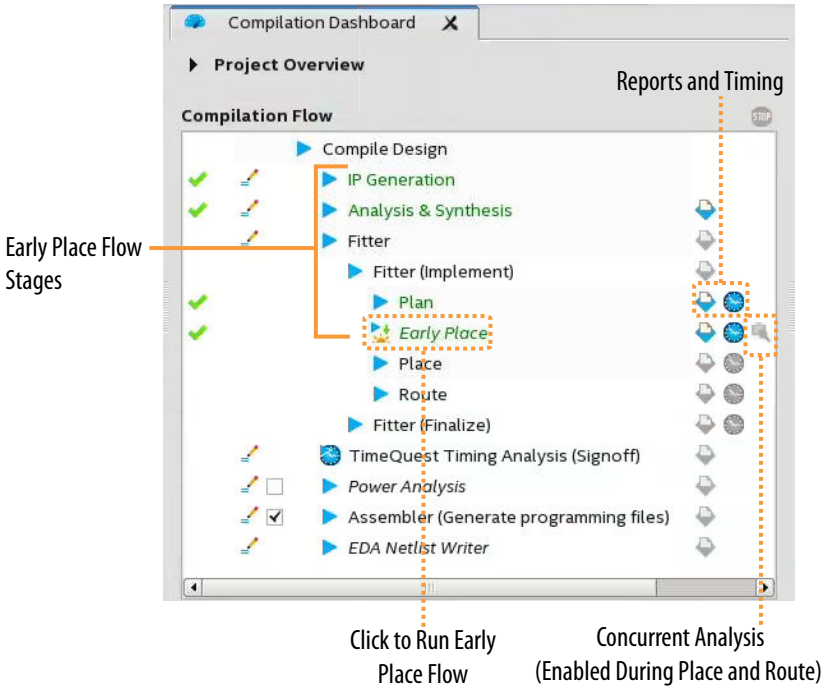


Because full compilation of a large design can be time consuming, use full compilation only when your design is ready for processing through all Compiler modules. During earlier stages of design iteration and debugging, and for large designs, it is more efficient to run and optimize Compiler modules individually, rather than running full compilation to obtain results. Running full compilation may also be suitable for one-click processing of a small design.

6.2.2 Early Place Flow

Early Place begins assigning core logic to device resources. Run **Early Place** to quickly view the effect of iterative floorplanning changes, without waiting for full placement or full compilation. The Compiler preserves a snapshot of the Early Place results. Following Early Place, click the **TimeQuest** icon to analyze Early Place timing. Optionally, adjust timing settings and constraints in TimeQuest before proceeding with compilation. Early Place runs automatically during Fitter processing if you enable **Settings ➤ Compiler Settings ➤ Fitter Settings (Advanced) ➤ Run Early Place During Compilation**.

Figure 57. Early Place Flow in Compilation Dashboard



If you run the Fitter (or **Place** or **Route** stages) without previously running **Early Place**, you can access the Early Place results while downstream Fitter stages are still running. Click the **Concurrent Analysis** icon on the Dashboard to analyze Early Place timing while the Fitter continues processing. You cannot modify timing constraints during concurrent analysis. However, stop compilation processing at any time, and then click the **TimeQuest** icon to modify constraints.

Note: Early Place does not run during full compilation by default. To enable Early Place during full compilation, click **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Fitter)** to modify the **Enable Early Place During Compilation** option.

6.2.3 Incremental Optimization Flow

The Quartus Prime Pro Edition supports incremental optimization at each stage of design compilation. In incremental optimization, you run and optimize each compilation stage independently before running the next compilation module in sequence. The Compiler preserves the results of each stage as a snapshot for

analysis. When you make changes to your design or constraints, the Compiler only runs stages impacted by the change. Following synthesis or any Fitter stage, view results and perform timing analysis. Modify design RTL or Compiler settings, as needed. Then, re-run synthesis or the Fitter and evaluate the results of these changes. Repeat this process until the module performance meets requirements. This flow maximizes the results at each stage, without waiting for full compilation results.

Figure 58. Incremental Optimization Flow

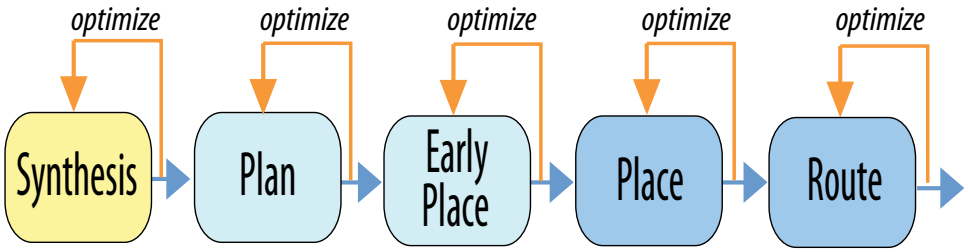


Table 33. Incremental Optimization at Fitter Stages

Fitter Stage	Incremental Optimization
Early Place	After this stage, the Chip Planner can display initial high-level placement of design elements. Use this information to guide your floorplanning decisions.
Place	After this stage, validate resource and logic utilization in the Compilation Reports, and review placement of design elements in the Chip Planner.
Route	After this stage, perform detailed setup and hold timing closure in TimeQuest, and view routing congestion via the Chip Planner.

6.3 Running Synthesis

Run design synthesis as part of a full compilation, or as an independent process. Before running synthesis, specify settings that control synthesis processing.

The Messages window dynamically displays processing information, warnings, or errors. Following Analysis and Synthesis processing, the Synthesis report provides detailed information about the synthesis of each design partition.

To run synthesis, follow these steps:

1. Create or open a Quartus Prime project with valid design files for compilation.
2. Before running synthesis, specify any of the following settings and constraints that impact synthesis:

- To specify options for the synthesis of Verilog HDL input files, click **Assignments ➤ Settings ➤ Verilog HDL Input**.
- To specify options for the synthesis of VHDL input files, click **Assignments ➤ Settings ➤ VHDL Input**.
- To specify options that affect compilation processing time, click **Assignments ➤ Settings ➤ Compilation Process Settings**.
- To specify advanced synthesis settings, click **Assignments ➤ Settings ➤ Compiler Settings**, and then click **Advanced Settings (Synthesis)**. Optionally, enable **Timing-Driven Synthesis** to account for timing constraints during synthesis.

3. To run synthesis, click **Synthesis** on the Compilation Dashboard.

Related Links

Synthesis Settings Reference on page 206

6.3.1 Preserve Registers During Synthesis

Quartus Prime synthesis minimizes gate count, merges redundant logic, and ensures efficient use of device resources. If you need to preserve specific registers through synthesis processing, you can specify any of the following entity-level assignments. Use **Preserve Resisters in Synthesis** or **Preserve Fan-Out Free Register Node** to allow Fitter optimization of the preserved registers. **Preserve Resisters** restricts Fitter optimization of the preserved registers. Specify synthesis preservation assignments by clicking **Assignments ➤ Assignment Editor**, in the `.qsf` file, or as synthesis attributes in your RTL.

Table 34. Synthesis Preserve Options

Assignment	Description	Allows Fitter Optimization?	Assignment Syntax
Preserve Resisters in Synthesis	Prevents removal of registers during synthesis. This settings does not affect retiming or other optimizations in the Fitter.	Yes	• `PRESERVE_REGISTER_SYN_ONLY ON|Off -to <entity> (.qsf)` • `preserve_syn_only` or `syn_preservesyn_only` (synthesis attributes)
Preserve Fan-Out Free Register Node	Prevents removal of assigned registers without fan-out during synthesis.	Yes	• `PRESERVE_REGISTER_FANOUT_FREE_NODE ON|Off -to <entity> (.qsf)` • `no_prune on` (synthesis attribute)
Preserve Resisters	Prevents removal and sequential optimization of assigned registers during synthesis. Sequential netlist optimizations can eliminate redundant registers and registers with constant drivers.	No	• `PRESERVE_REGISTER ON|Off -to <entity> (.qsf)` • `preserve`, `syn_preserve`, or `keep on` (synthesis attributes)

6.3.2 Enabling Timing-Driven Synthesis

Timing-driven synthesis directs the Compiler to account for your timing constraints during synthesis. Timing-driven synthesis runs initial timing analysis to obtain netlist timing information. Synthesis then focuses performance efforts on timing-critical design elements, while optimizing non-timing-critical portions for area.

Timing-driven synthesis preserves timing constraints, and does not perform optimizations that conflict with timing constraints. Timing-driven synthesis may increase the number of required device resources. Specifically, the number of adaptive look-up tables (ALUTs) and registers may increase. The overall area can increase or decrease. Runtime and peak memory use increases slightly.

Quartus Prime Pro Edition runs timing-driven synthesis by default. To enable or disable this option manually, click **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Synthesis)**.

Related Links

- Running Synthesis on page 189
- Synthesis Language Support on page 201

6.3.3 Enabling Multi-Processor Compilation

The Compiler can detect and use multiple processors to reduce total compilation time. You can control the number of processors the Compiler uses. The Quartus Prime software can use up to 16 processors to run algorithms in parallel. The Compiler uses parallel compilation by default. To reserve some processors for other tasks, specify a maximum number of processors that the software uses.

You can reduce the compilation time by up to 10% on systems with two processing cores and by up to 20% on systems with four cores. When running timing analysis independently, two processors can reduce the time timing analysis time by an average of 10%. This reduction can reach an average of 15% when using four processors.

The Quartus Prime software does not necessarily use all the processors that you specify during a given compilation. Additionally, the software never uses more than the specified number of processors, enabling you to work on other tasks on your computer without it becoming slow or less responsive. The use of multiple processors does not affect the quality of the fit. For a given Fitter seed and given **Maximum processors allowed** setting on a specific design, the fit is exactly the same and deterministic, regardless of the target machine and the number of available processors. Different **Maximum processors allowed** specifications produce different results of the same quality. The impact is similar to changing the Fitter seed setting.

To enable multiprocessor compilation, follow these steps:

1. Open or create a Quartus Prime project.
2. To enable multiprocessor compilation, click **Assignments ➤ Settings ➤ Compilation Process Settings**.
3. Under **Parallel compilation**, specify options for the number of processors the Compiler uses.
4. View detailed information about processor in the Parallel Compilation report following compilation.

To specify the number of processors for compilation at the command line, use the following Tcl command in your script:

```
set_global_assignment -name NUM_PARALLEL_PROCESSORS <value>
```

In this case, <*value*> is an integer from 1 to 16.

If you want the Quartus Prime software to detect the number of processors and use all the processors for the compilation, include the following Tcl command in your script:

```
set_global_assignment -name NUM_PARALLEL_PROCESSORS ALL
```

Note: The Compiler detects Intel Hyper-Threading as a single processor. If your system includes a single processor with Intel Hyper-Threading, set the number of processors to one. Do not use the Intel Hyper-Threading feature for Quartus Prime compilations.

6.3.4 Synthesis Reports

The Compilation Report window opens automatically during compilation processing. The Report window displays detailed synthesis results for each partition in the current project revision.

Figure 59. Synthesis Reports

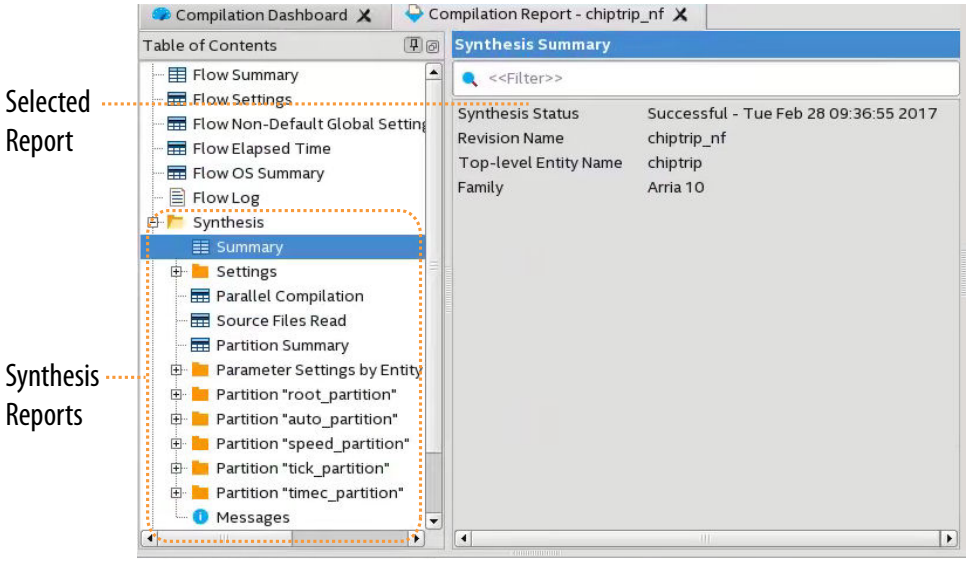


Table 35. Synthesis Reports (Design Dependent)

Generated Report	Description
Summary	Shows summary information about synthesis, such as the status, date, software version, entity name, device family, timing model status, and various types of logic utilization.
Synthesis Settings	Lists the values of all synthesis settings during design processing.
Parallel Compilation	Lists specifications for any use of parallel processing during synthesis.

continued...

Generated Report	Description
Resource Utilization By Entity	Lists the quantity of all types of logic usage for each entity in design synthesis.
Multiplexer Restructuring Statistics	Provides statistics for the amount of multiplexer restructuring that synthesis performs.
IP Cores Summary	Lists details about each IP core instance in design synthesis. Details include IP core name, vendor, version, license type, entity instance, and IP include file.
Synthesis Source Files Read	Lists details about all source files in design synthesis. Details include file path, file type, and any library information.
Resource Usage Summary for Partition	Lists the quantity of all types of logic usage for each design partition in design synthesis.
RAM Summary for Partition	Lists RAM usage details for each design partition in design synthesis. Details include the name, type, mode, and density.
Register Statistics	Lists the number of registers using various types of global signals.
Synthesis Messages	Lists all information, warning, and error messages that report conditions observed during the Analysis & Synthesis process.

6.4 Running the Fitter

The Compiler's Fitter module performs design place and route. Run all stages of the Fitter automatically as part of a full design compilation, or run the Fitter or any Fitter stage independently after design synthesis. Before running the Fitter, specify settings that affect design fitting.

1. Specify initial Fitter constraints:
   a. To assign device I/O pins, click **Assignments ➤ Pin Planner**.
   b. To assign device periphery, clocks, and I/O interfaces, click **Tools ➤ BluePrint Platform Designer**.
   c. To constrain logic placement regions, click **Tools ➤ Chip Planner**.
   d. To specify general performance, power, or logic usage focus for fitting, click **Assignments ➤ Settings ➤ Compiler Settings**.
   e. To fine-tune place and route with advanced Fitter options, click **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Fitter)**
2. To run one or more stages of the Fitter, click any of the following commands on the Compilation Dashboard:
   - To begin device periphery placement and routing, click **Plan**.
   - To run early placement, click **Early Place**.
   - To fully place design logic, click **Place**.
   - To fully route the design, click **Route**.
   - To run the Implement flow (Plan, Place, and Route stages), click **Fitter (Implement)**.
   - To run the Finalize flow (Plan, Early Place, Place, Route, and Finalize stages), click **Fitter (Finalize)**.
   - To run all Fitter stages in sequence, click **Fitter**.

Related Links

Fitter Settings Reference on page 213

6.4.1 Fitter Stage Commands

Launch Fitter processes from the **Processing** menu or Compilation Dashboard.

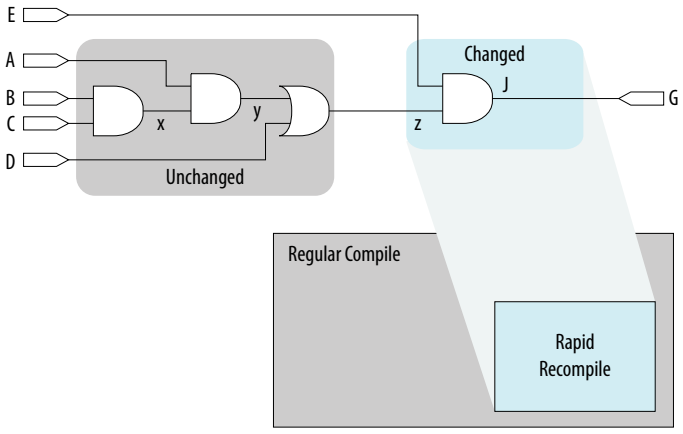
Table 36. Fitter Stage Commands

Command	Description
Fitter (Implement)	Runs the Plan, Early Place, Place, and Route stages. Click the adjacent **TimeQuest** icon after this stage to analyze the subset of timing corners needed for timing closure.
Start Fitter (Plan)	Loads synthesized periphery placement data and constraints, and assigns periphery elements to device I/O resources. After this stage, you can run post-Plan timing analysis to verify timing constraints, check periphery timing, and validate cross-clock timing windows. This command creates the Planned snapshot.
Start Fitter (Early Place)	Begins assigning core design logic to device resources. After this stage, the Chip Planner can display initial high-level placement of design elements. Use this information to guide your floorplanning decisions. This command creates the Early Placed snapshot. Early Place does not run during the full compilation flow.
Start Fitter (Place)	Completes assignment of core design logic placement to device resources. This command creates the Placed snapshot.
Start Fitter (Route)	Performs core routing. This stage creates a fully routed design to validate delay chain settings and analyze routing resources. After this stage, perform detailed setup and hold timing closure in The TimeQuest Timing Analyzer and view routing congestion via the Chip Planner. This command creates the Routed snapshot.
Start Fitter (Finalize)	Finalizes the place and route process after timing closure. This stage converts unneeded tiles from High Speed to Low Power. This command creates the Final snapshot.

6.4.2 Running Rapid Recompile

During Rapid Recompile the Compiler reuses previous synthesis and fitting results whenever possible, and does not reprocess unchanged design blocks. Use Rapid Recompile to reduce timing variations and the total recompilation time after making small design changes.

Figure 60. Rapid Recompile

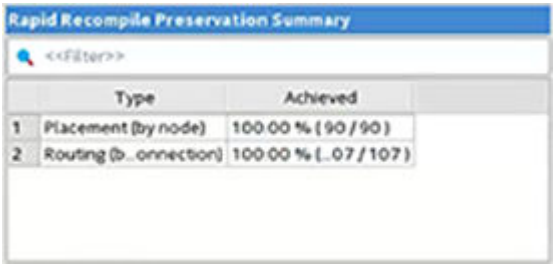


To run Rapid Recompile, follow these steps:

1. Open or create a Quartus Prime project.
2. To start Rapid Recompile following an initial compilation (or after running the Route stage of the Fitter), click **Processing ➤ Start ➤ Start Rapid Recompile**. Rapid Recompile implements the following types of design changes without full recompilation:
   - Changes to nodes tapped by the Signal Tap Logic Analyzer
   - Changes to combinational logic functions
   - Changes to state machine logic (for example, new states, state transition changes)
   - Changes to signal or bus latency or addition of pipeline registers
   - Changes to coefficients of an adder or multiplier
   - Changes register packing behavior of DSP, RAM, or I/O
   - Removal of unnecessary logic
   - Changes to synthesis directives

   The Rapid Recompile Preservation Summary report provides detailed information about the percentage of preserved compilation results.

Figure 61. Rapid Recompile Preservation Summary



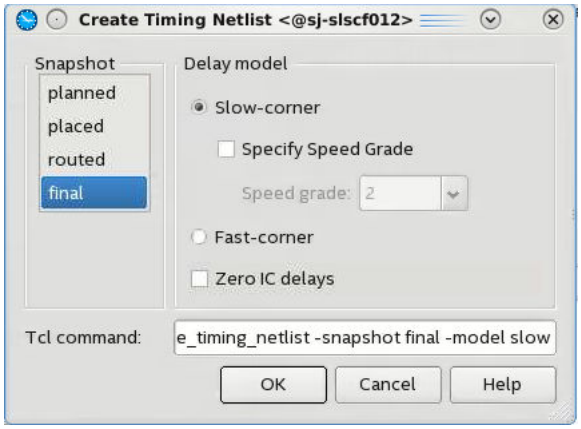
6.4.3 Analyzing Fitter Stage Timing

Run timing analysis on the results of any Fitter stage to evaluate performance before running the next Compilation module. After running any stage of the Fitter, click the adjacent **TimeQuest** icon in the Compilation Dashboard to load that snapshot's timing netlist in the TimeQuest Timing Analyzer.

Follow these steps to analyze timing for a specific Fitter snapshot:

1. To run any stage of the Fitter, click **Plan**, **Early Place**, **Place**, or **Route** on the Compilation Dashboard.
2. On the Compilation Dashboard, click the **Timing Analyzer** icon adjacent to the Fitter stage. The **Create Timing Netlist** dialog box loads the corresponding stage snapshot.
3. Click **OK**. The Timing Analyzer loads the timing netlist and delay model for analysis.
4. At any time, click **Netlist ➤ Create Timing Netlist** to load another **Snapshot** or **Delay model**.
5. Analyze the timing of the Fitter stage in the Timing Analyzer. Determine if the stage results meet your requirements.

Figure 62. Create Timing Netlist



Related Links

The Quartus Prime TimeQuest Timing Analyzer

6.4.4 Enabling Physical Synthesis Optimization

Physical synthesis optimization improves circuit performance by performing combinational and sequential optimization and register duplication.

To enable physical synthesis options, follow these steps:

1. Click **Assignments ➤ Settings ➤ Compiler Settings**.
2. To enable retiming, combinational optimization, and register duplication, click **Advanced Settings (Fitter)**. Next, enable **Physical Synthesis**.
3. View physical synthesis results in the **Netlist Optimizations** report.

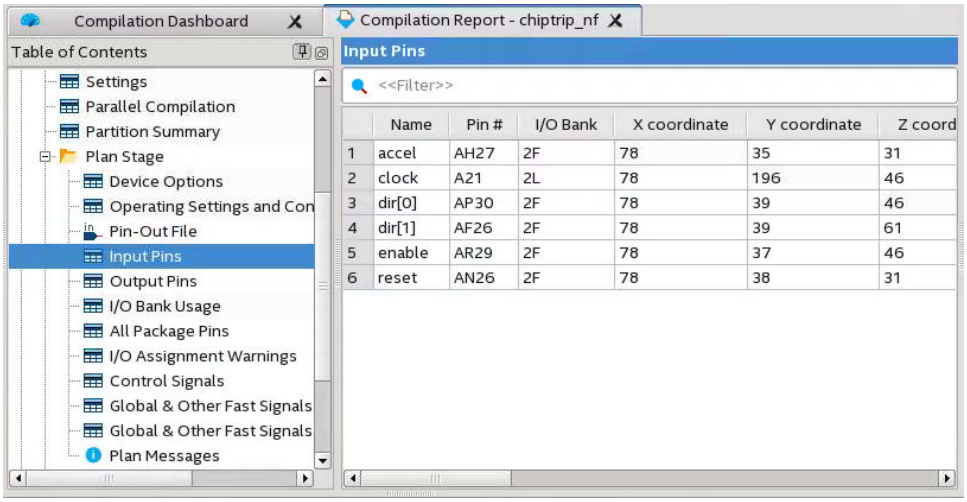
6.4.5 Viewing Fitter Reports

The Fitter generates detailed reports and messages for each stage of place and route. The Fitter Summary reports basic information about the Fitter run, such as date, software version, device family, timing model, and logic utilization.

6.4.5.1 Plan Stage Reports

The Fitter generates reports for each stage. The stage reports provide information for analysis and optimization of each Fitter stage. The Plan stage reports describe the I/O, interface, and control signals discovered during the periphery planning stage of the Fitter.

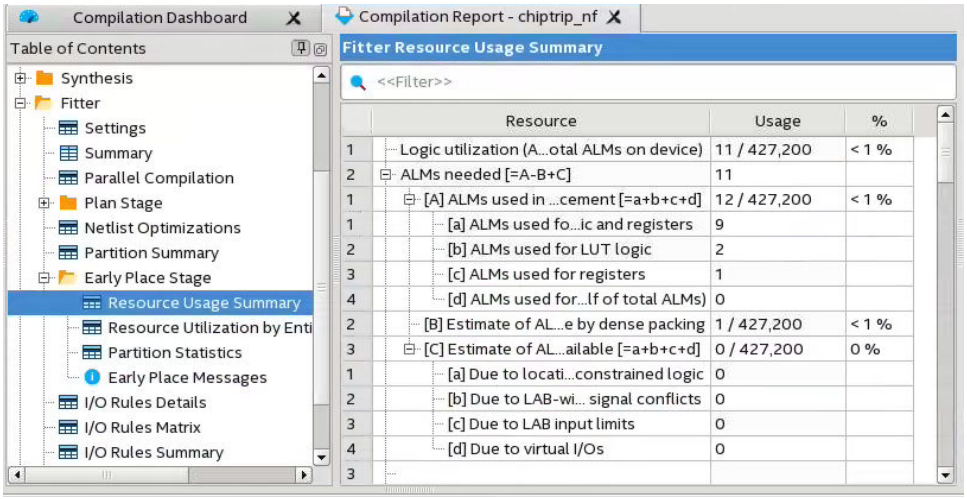
Figure 63. Plan Stage Reports



6.4.5.2 Early Place Stage Reports

During Early Place the Fitter begins assigning core design logic to device resources.

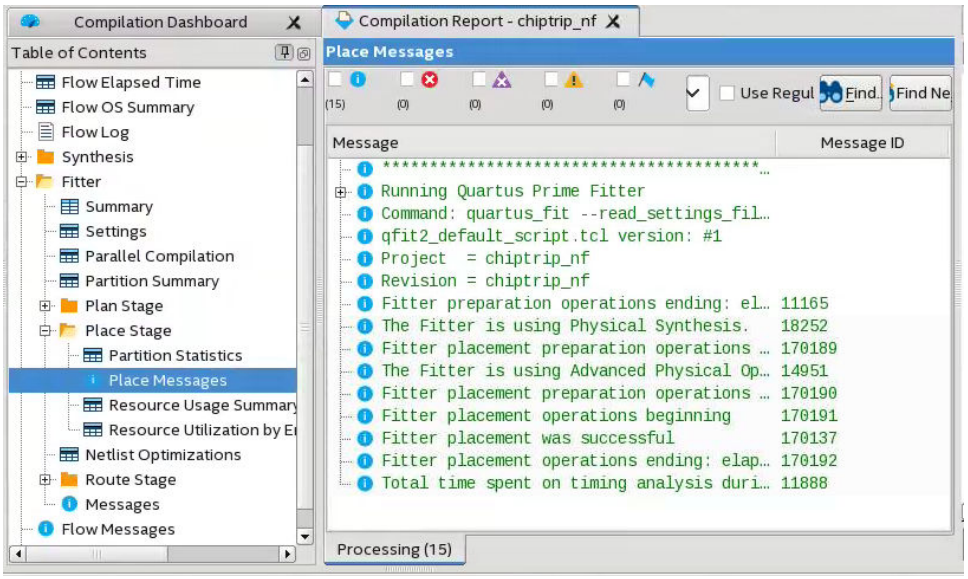
Figure 64. Early Place Stage Reports



6.4.5.3 Place Stage Reports

The Place stage reports describe all device resources the Fitter allocates during logic placement. The report details include the type, number, and overall percentage of each resource type.

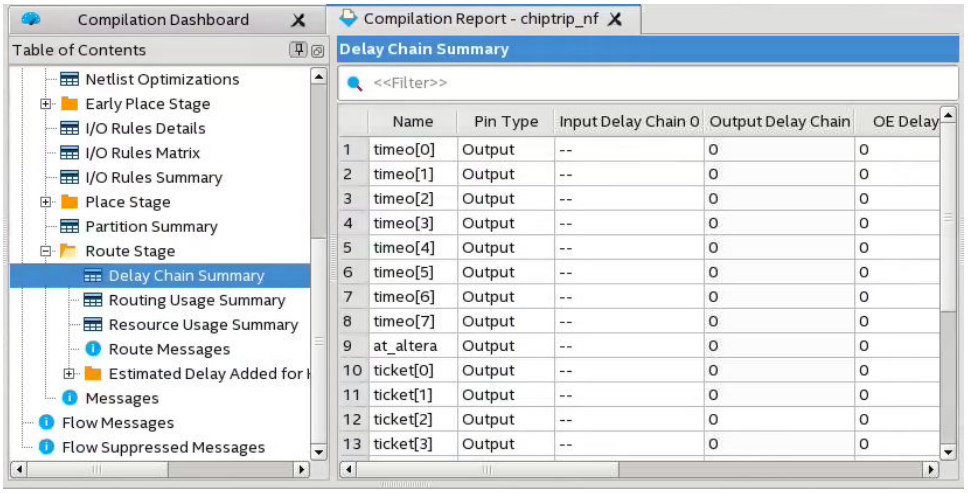
Figure 65. Place Stage Reports

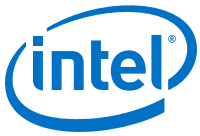


6.4.5.4 Route Stage Reports

The Route stage reports describe all device resources that the Fitter allocates during routing. Details include the type, number, and overall percentage of each resource type.

Figure 66. Route Stage Reports





6.4.5.5 Finalize Stage Reports

The Finalize stage reports describe final placement and routing operations, including:

- Delay chain summary information

6.5 Running Full Compilation

Use these steps to run a full compilation of a Quartus Prime design project. A full compilation includes IP Generation, Analysis & Synthesis, Fitter, TimeQuest, and any optional modules that you enable in the Compilation Dashboard.

1. Before running a full compilation, specify any of the following project settings:
   - To specify the target FPGA device or development kit, click **Assignments ➤ Device**.
   - To specify device and pin options for the target FPGA device, click **Assignments ➤ Device ➤ Device and Pin Options**.
   - To specify options that affect compilation processing time and netlist preservation, click **Assignments ➤ Settings ➤ Compilation Process Settings**.
   - To specify synthesis algorithm and other **Advanced Settings** for synthesis and fitting, click **Assignments ➤ Settings ➤ Compiler Settings**.
   - To specify required timing conditions for proper operation of your design, click **Tools ➤ TimeQuest Timing Analyzer**.
2. Specify interface and I/O constraints:
   - To plan placement of device periphery interfaces and clocking, click **Tools ➤ BluePrint Platform Designer**.
   - To edit, validate, or export pin assignments, click **Assignments ➤ Pin Planner**.
3. To run full compilation, click **Processing ➤ Start Compilation**.

   Click **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Fitter)** to access Fitter settings

   Note: Early Place does not run during full compilation by default. To enable Early Place during full compilation, click **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Fitter)** to modify the **Run Early Place during compilation** option.

Related Links

- BluePrint Design Planning
- The TimeQuest Timing Analyzer
- Managing Device I/O Pins

6.6 Generating Programming Files

The Compiler's Assembler module generates device programming files. Run the Assembler to generate device programming files following successful design place and route.

1. Before running the Assembler, specify settings to customize programming file generation. Click **Assignments ➤ Device ➤ Device & Pin Options** to enable or disable generation of optional programming files.
2. To generate device programming files, click **Processing ➤ Start ➤ Start Assembler**, or click **Assembler** on the Compilation Dashboard. The Compiler confirms that prerequisite modules are complete, and launches the Assembler module to generate the programming files that you specify. The Messages window dynamically displays processing information, warnings, or errors. After Assembler processing,

After running the Assembler, the Compilation report provides detailed information about programming file generation, including programming file Summary and Encrypted IP information.

Figure 67. Assembler Reports

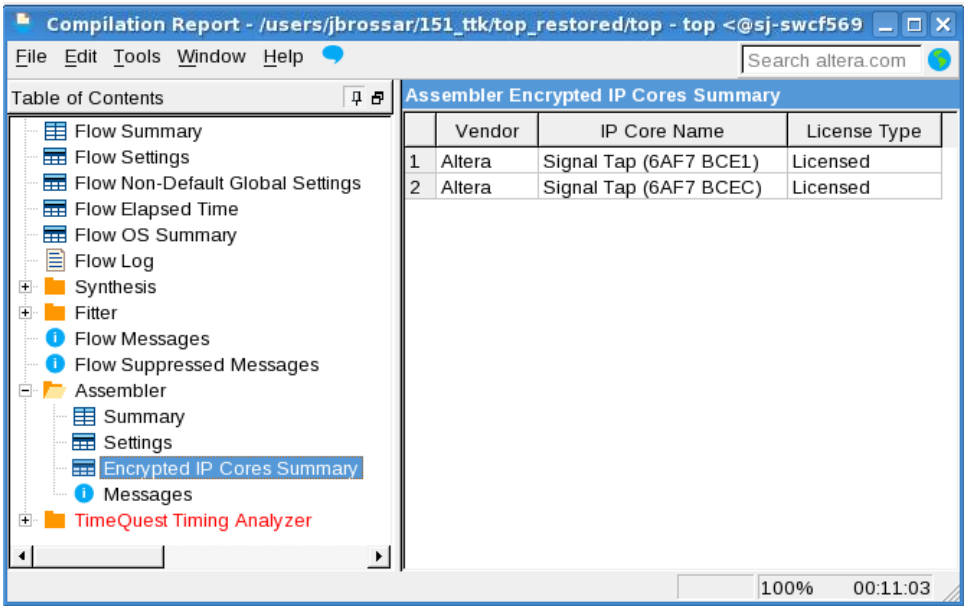
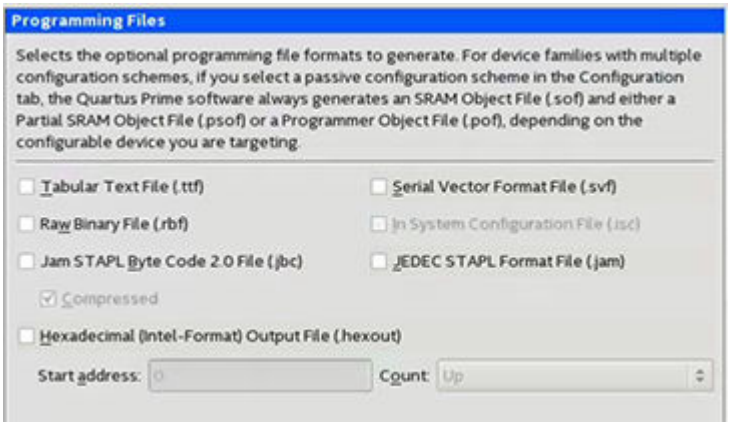


Figure 68. Device & Pin Options



Related Links

Programming Intel FPGA Devices

6.7 Synthesis Language Support

The Quartus Prime software synthesizes standard Verilog HDL, VHDL, and SystemVerilog design files.

6.7.1 Verilog and SystemVerilog Synthesis Support

Quartus Prime synthesis supports the following Verilog HDL language standards:

- Verilog-1995 (IEEE Standard 1364-1995)
- Verilog-2001 (IEEE Standard 1364-2001)
- SystemVerilog-2005 (IEEE Standard 1800-2005)
- SystemVerilog-2009 (IEEE Standard 1800-2009)

The following important guidelines apply to Quartus Prime synthesis of Verilog HDL and SystemVerilog:

- The Compiler uses the Verilog-2001 standard by default for files with an extension of `.v`, and the SystemVerilog standard for files with the extension of `.sv`.
- If you use scripts to add design files, you can use the `-HDL_VERSION` command to specify the HDL version for each design file.
- Compiler support for Verilog HDL is case sensitive in accordance with the Verilog HDL standard.
- The Compiler supports the compiler directive `` `define ``, in accordance with the Verilog HDL standard.
- The Compiler supports the `include` compiler directive to include files with absolute paths (with either "/" or "\" as the separator), or relative paths.
- When searching for a relative path, the Compiler initially searches relative to the project directory. If the Compiler cannot find the file, the Compiler next searches relative to all user libraries. Finally, the Compiler searches relative to the current file's directory location.
- Quartus Prime Pro Edition synthesis searches for all modules or entities earlier in the synthesis process than other Quartus software tools. This earlier search produces earlier syntax errors for undefined entities than other Quartus software tools.

Related Links

- The Quartus Prime TimeQuest Timing Analyzer
- Recommended Design Practices
- Recommended HDL Coding Styles

6.7.1.1 Verilog HDL Input Settings (Settings Dialog Box)

Click **Assignments ➤ Settings ➤ Verilog HDL Input** to specify options for the synthesis of Verilog HDL input files.

Figure 69. Verilog HDL Input Settings Dialog Box

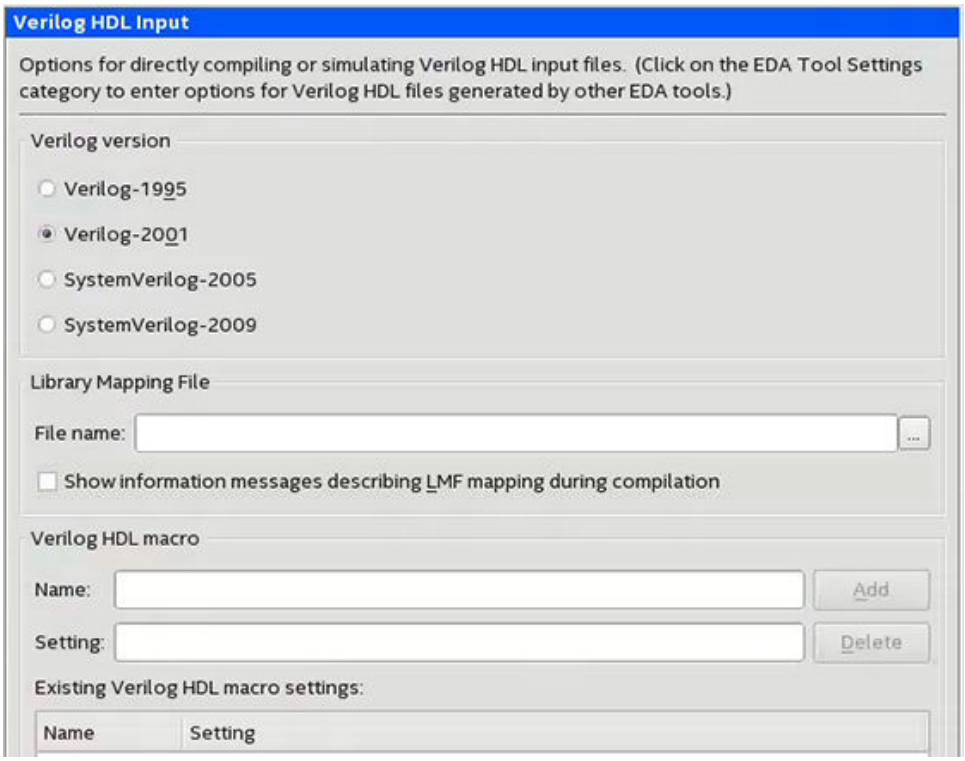


Table 37. Verilog HDL Input Settings

Setting	Description
Verilog Version	Directs synthesis to process Verilog HDL input design files using the specified standard. You can select any of the supported language standards to match your Verilog HDL files or SystemVerilog design files.
Library Mapping File	Allows you to optionally specify a provided Library Mapping File (`.lmf`) for use in synthesizing Verilog HDL files that contain non-Intel functions mapped to IP cores. You can specify the full path name of the LMF in the **File name** box.
Verilog HDL Macro	Verilog HDL macros are pre-compiler directives which can be added to Verilog HDL files to define constants, flags, or other features by **Name** and **Setting**. Macros that you add appear in the **Existing Verilog HDL macro settings** list.

6.7.1.2 Design Libraries

By default, the Compiler processes all design files into one or more libraries.

- When compiling a design instance, the Compiler initially searches for the entity in the library associated with the instance (which is the work library if you do not specify any library).
- If the Compiler cannot locate the entity definition, the Compiler searches for a unique entity definition in all design libraries.
- If the Compiler finds more than one entity with the same name, the Compiler generates an error. If your design uses multiple entities with the same name, you must compile the entities into separate libraries.

6.7.1.3 Verilog HDL Configuration

Verilog HDL configuration is a set of rules that specify the source code for particular instances. Verilog HDL configuration allows you to perform the following tasks:

- Specify a library search order for resolving cell instances (as does a library mapping file).
- Specify overrides to the logical library search order for specified instances.
- Specify overrides to the logical library search order for all instances of specified cells.

Related Links

Configuration Syntax

6.7.1.3.1 Hierarchical Design Configurations

A design can have more than one configuration. For example, you can define a configuration that specifies the source code you use in particular instances in a sub-hierarchy, and then define a configuration for a higher level of the design.

For example, suppose a subhierarchy of a design is an eight-bit adder, and the RTL Verilog code describes the adder in a logical library named `rtllib`. The gate-level code describes the adder in the `gatelib` logical library. If you want to use the gate-level code for the 0 (zero) bit of the adder and the RTL level code for the other seven bits, the configuration might appear as follows:

Example 63. Gate-level code for the 0 (zero) bit of the adder

```
config cfg1;
design aLib.eight_adder;
default liblist rtllib;
instance adder.fulladd0 liblist gatelib;
endconfig
```

If you are instantiating this eight-bit adder eight times to create a 64-bit adder, use configuration `cfg1` for the first instance of the eight-bit adder, but not in any other instance. A configuration that performs this function is shown below:

Example 64. Use configuration `cfg1` for first instance of eight-bit adder

```
config cfg2;
design bLib.64_adder;
default liblist bLib;
instance top.64add0 use work.cfg1:config;
endconfig
```

Note: The name of the unbound module may be different from the name of the cell that is bounded to the instance.

6.7.1.4 Initial Constructs and Memory System Tasks

The Quartus Prime software infers power-up conditions from the Verilog HDL `initial` constructs. The Quartus Prime software also creates power-up settings for variables, including RAM blocks. If the Quartus Prime software encounters non-synthesizable constructs in an `initial` block, it generates an error.

To avoid such errors, enclose non-synthesizable constructs (such as those intended only for simulation) in `translate_off` and `translate_on` synthesis directives. Synthesis of initial constructs enables the power-up state of the synthesized design to match the power-up state of the original HDL code in simulation.

Note: Initial blocks do not infer power-up conditions in some third-party EDA synthesis tools. If you convert between synthesis tools, you must set your power-up conditions correctly.

Quartus Prime synthesis supports the `$readmemb` and `$readmemh` system tasks to initialize memories.

Example 65. Verilog HDL Code: Initializing RAM with the readmemb Command

```
reg [7:0] ram[0:15];
initial
begin
$readmemb("ram.txt", ram);
end
```

When creating a text file to use for memory initialization, specify the address using the format @<*location*> on a new line, and then specify the memory word such as `110101` or `abcde` on the next line.

The following example shows a portion of a Memory Initialization File (`.mif`) for the RAM.

Example 66. Text File Format: Initializing RAM with the readmemb Command

```
@0
00000000
@1
00000001
@2
00000010
…
@e
00001110
@f
00001111
```

Related Links

- Translate Off and On / Synthesis Off and On
- Incremental Compilation for Hierarchical and Team-Based Design

6.7.1.5 Verilog HDL Macros

The Quartus Prime software fully supports Verilog HDL macros, which you can define with the `'define` compiler directive in your source code. You can also define macros in the Quartus Prime software or on the command line.

6.7.2 VHDL Synthesis Support

Quartus Prime synthesis supports the following VHDL language standards.

- VHDL 1987 (IEEE Standard 1076-1987)
- VHDL 1993 (IEEE Standard 1076-1993)
- VHDL 2008 (IEEE Standard 1076-2008)

The Quartus Prime Compiler uses the VHDL 1993 standard by default for files that have the extension `.vhdl` or `.vhd`.

Note: The VHDL code samples follow the VHDL 1993 standard.

Related Links

Migrating to Quartus Prime Pro Edition

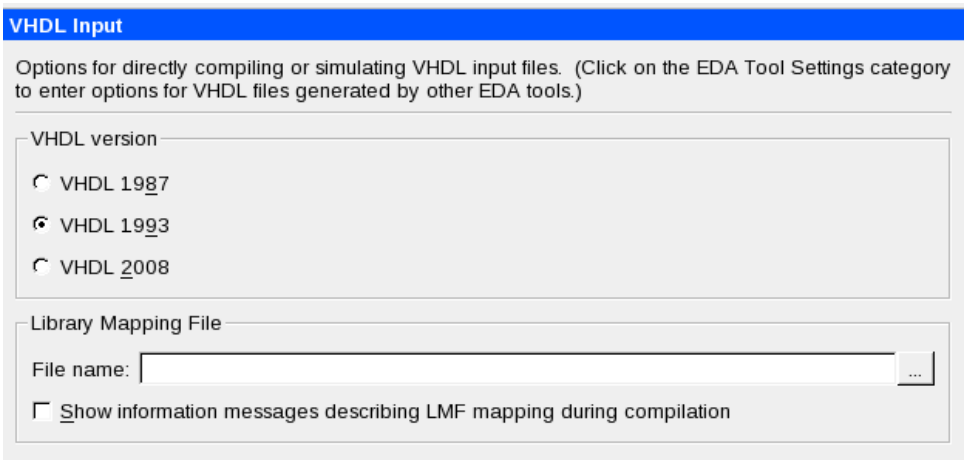
6.7.2.1 VHDL Input Settings (Settings Dialog Box)

Click **Assignments ➤ Settings ➤ VHDL Input** to specify options for the synthesis of VHDL input files.

Table 38. VHDL Input Settings

Setting	Description
VHDL Version	Specifies the VHDL standard for use during synthesis of VHDL input design files. Select the language standards that corresponds with the VHDL files.
Library Mapping File	Specifies a Library Mapping File (`.lmf`) for use in synthesizing VHDL files that contain IP cores. Specify the full path name of the LMF in the **File name** box.

Figure 70. VHDL Input Settings Dialog Box



6.7.2.2 VHDL Standard Libraries and Packages

The Quartus Prime software includes the standard IEEE libraries and several vendor-specific VHDL libraries. The IEEE library includes the standard VHDL packages `std_logic_1164`, `numeric_std`, `numeric_bit`, and `math_real`.

The STD library is part of the VHDL language standard and includes the packages `standard` (included in every project by default) and `textio`. For compatibility with older designs, the Quartus Prime software also supports the following vendor-specific packages and libraries:

- Synopsys packages such as `std_logic_arith` and `std_logic_unsigned` in the IEEE library.
- Mentor Graphics® packages such as `std_logic_arith` in the ARITHMETIC library.
- Primitive packages `altera_primitives_components` (for primitives such as GLOBAL and DFFE) and `maxplus2` in the ALTERA library.
- IP core packages `altera_mf_components` in the ALTERA_MF library for specific IP cores including LCELL. In addition, `lpm_components` in the LPM library for library of parameterized modules (LPM) functions.

Note: Import component declarations for primitives such as GLOBAL and DFFE from the `altera_primitives_components` package and not the `altera_mf_components` package.

6.7.2.3 VHDL wait Constructs

The Quartus Prime software supports one VHDL `wait until` statement per process block. However, the Quartus Prime software does not support other VHDL wait constructs, such as `wait for` and `wait on` statements, or processes with multiple `wait` statements.

Example 67. VHDL `wait until` construct example

```
architecture dff_arch of ls_dff is
begin
output: process begin
wait until (CLK'event and CLK='1');
Q <= D;
Qbar <= not D;
end process output;
end dff_arch;
```

6.8 Synthesis Settings Reference

This section provides a reference to all synthesis settings. Use these settings to customize synthesis processing for your design goals.

6.8.1 Optimization Modes

The following options direct the focus of Compiler optimization efforts during synthesis. The settings affect synthesis and fitting.

Table 39. Optimization Modes (Compiler Settings Page)

Optimization Mode	Description
Balanced (Normal Flow)	Optimizes synthesis for balanced implementation that respects timing constraints.
Performance (High effort - increases runtime)	Makes high effort to optimize synthesis for speed performance. High effort increases synthesis run time.
Performance (Aggressive - increases runtime and area)	Makes aggressive effort to optimize synthesis for speed performance. Aggressive effort increases synthesis run time and device resource use.
Power (High effort - increases runtime)	Makes high effort to optimize synthesis for low power. High effort increases synthesis run time.
Power (Aggressive - increases runtime, reduces performance)	Makes aggressive effort to optimize synthesis for low power. Aggressive effort increases synthesis time and reduces speed performance.
Area (Aggressive - reduces performance)	Makes aggressive effort to reduce the device area required to implement the design.

6.8.2 Prevent Register Retiming

Enable the **Prevent register retiming** option if you want to globally prevent automatic retiming of registers for design performance improvement. When disabled, the Compiler automatically performs register retiming optimizations that move combinational logic across register boundaries. The Compiler maintains the overall logic of the design component, and also balances the datapath delays between each register. Optionally, assign **Allow Register Retiming** to any design entity or instance to override **Prevent register retiming** for specific portions of the design. Click **Assignments ➤ Assignment Editor** to specify entity- and instance-level assignments, or use the following syntax to make the assignment in the `.qsf` directly.

Example 68. Disable register retiming for entity abc

```
set_global_assignment –name ALLOW_REGISTER_RETIMING ON
set_instance_assignment –name ALLOW_REGISTER_RETIMING OFF –to “abc|”
set_instance_assignment –name ALLOW_REGISTER_RETIMING ON –to “abc|def|”
```

Example 69. Disable register retiming for the whole design, except for registers in entity abc

```
set_global_assignment –name ALLOW_REGISTER_RETIMING OFF
set_instance_assignment –name ALLOW_REGISTER_RETIMING ON –to “abc|”
set_instance_assignment –name ALLOW_REGISTER_RETIMING OFF –to “abc|def|”
```

6.8.3 Advanced Synthesis Settings

The following section is a quick reference of all Advanced Synthesis Settings. Click **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Synthesis)** to modify these settings.

Table 40. Advanced Synthesis Settings (1 of 13)

Option	Description
Allow Any RAM Size for Recognition	Allows the Compiler to infer RAMs of any size, even if the RAMs do not meet the current minimum requirements.
Allow Any ROM Size for Recognition	Allows the Compiler to infer ROMs of any size even if the ROMs do not meet the design's current minimum size requirements.
Allow Any Shift Register Size for Recognition	Allows the Compiler to infer shift registers of any size even if they do not meet the design's current minimum size requirements.
Allow Register Duplication	Controls whether the Compiler duplicates registers to improve design performance. When enabled, the Compiler performs optimization that creates a second copy of a register and move a portion of its fan-out to this new node. This technique improves routability and/or reduces the total routing wire required to route a net with many fan-outs. If you disable this option, retiming of registers is also disabled.
Allow Register Merging	Controls whether the Compiler removes (merges) identical registers. When enabled, in cases where two registers generate the same logic, the Compiler may delete one register and fan-out the remaining register to the deleted register's destinations. This option is useful if you wish to prevent the Compiler from removing duplicate registers that you have used deliberately. When disabled, retiming optimizations are also disabled.
Allow Shift Register Merging Across Hierarchies	Allows the Compiler to take shift registers from different hierarchies of the design and put the registers in the same RAM.
Allow Synchronous Control Signals	Allows the Compiler to utilize synchronous clear and/or synchronous load signals in normal mode logic cells. Turning on this option helps to reduce the total number of logic cells used in the design, but can negatively impact the fitting. This negative impact occurs because all the logic cells in a LAB share synchronous control signals.

Table 41. Advanced Synthesis Settings (2 of 13)

Option	Description
Analysis & Synthesis Message Level	Specifies the type of Analysis & Synthesis messages the Compiler display. **Low** displays only the most important Analysis & Synthesis messages. **Medium** displays most messages, but hides the detailed messages. **High** displays all messages.
Auto Carry Chains	Allows the Compiler to create carry chains automatically by inserting CARRY_SUM buffers into the design. The **Carry Chain Length** option controls the length of the chains. When this option is off, the Compiler ignores CARRY buffers, but CARRY_SUM buffers are unaffected. The Compiler ignores the **Auto Carry Chains** option if you select **Product Term** or **ROM** as the setting for the **Technology Mapper** option.
Auto Clock Enable Replacement	Allows the Compiler to locate logic that feeds a register and move the logic to the register's clock enable input port.
Auto DSP Block Replacement	Allows the Compiler to find a multiply-accumulate function or a multiply-add function that can be replaced with a DSP block.
Auto Gated Clock Conversion	Automatically converts gated clocks to use clock enable pins. Clock gating logic can contain AND, OR, MUX, and NOT gates. Turning on this option may increase memory use and overall run time. You must use the TimeQuest Timing Analyzer for timing analysis, and you must define all base clocks in Synopsys Design Constraints (.sdc) format.

Table 42. Advanced Synthesis Settings (3 of 13)

Option	Description
Auto Open-Drain Pins	Allows the Compiler to automatically convert a tri-state buffer with a strong low data input into the equivalent open-drain buffer.
Auto RAM Replacement	Allows the Compiler to identify sets of registers and logic that it can replace with the altsyncram or the lpm_ram_dp IP core. Turning on this option may change the functionality of the design.

continued...

Option	Description
Auto ROM Replacement	Allows the Compiler to identify logic that it can replace with the altsyncram or the lpm_rom IP core. Turning on this option may change the power-up state of the design.
Auto Resource Sharing	Allows the Compiler to share hardware resources among many similar, but mutually exclusive, operations in your HDL source code. If you enable this option, the Compiler merges compatible addition, subtraction, and multiplication operations. Merging operations may reduce the area your design requires. Because resource sharing introduces extra muxing and control logic on each shared resource, it may negatively impact the final f_{MAX} of your design.
Auto Shift Register Placement	Allows the Compiler to find a group of shift registers of the same length that are replaceable with the altshift_taps IP core. The shift registers must all use the same clock and clock enable signals. The registers must not have any other secondary signals. The registers must have equally spaced taps that are at least three registers apart.
Automatic Parallel Synthesis	Option to enable/disable automatic parallel synthesis. Use this option to speed up synthesis compile time by using multiple processors when available.

Table 43. Advanced Synthesis Settings (4 of 13)

Option	Description
Block Design Naming	Specifies the naming scheme for the block design. The Compiler ignores the option if you assign the option to anything other than a design entity.
Carry Chain Length	Specifies the maximum allowable length of a chain of both user-entered and Compiler-synthesized CARRY_SUM buffers. The Compiler breaks carry chains that exceed this length into separate chains.
Clock MUX Protection	Causes the multiplexers in the clock network to decompose to 2-to-1 multiplexer trees. The Compiler protects these trees from merging with, or transferring to, other logic. This option helps the TimeQuest Timing Analyzer to analyze clock behavior.
Create Debugging Nodes for IP Cores	Makes certain nodes (for example, important registers, pins, and state machines) visible for all the IP cores in a design. Use IP core nodes to effectively debug the IP core. This technique is effective when using the IP core with the Signal Tap Logic Analyzer. The Node Finder, using Signal Tap Logic Analyzer filters, displays all the nodes that Analysis & Synthesis makes visible. When making the debugging nodes visible, Analysis & Synthesis can change the f_{MAX} and number of logic cells in IP cores.
DSP Block Balancing	Allows you to control the conversion of certain DSP block slices during DSP block balancing.

Table 44. Advanced Synthesis Settings (5 of 13)

Option	Description
Disable DSP Negate Inferencing	Allows you to specify whether to use the negate port on an inferred DSP block.
Disable Register Merging Across Hierarchies	Specifies whether the Compiler allows merging of registers that are in different hierarchies if their inputs are the same.
Enable State Machines Inference	Allows the Compiler to infer state machines from VHDL or Verilog HDL design files. The Compiler optimizes state machines to reduce area and improve performance. If set to Off, the Compiler extracts and optimizes state machines in VHDL or Verilog HDL design files as regular logic.
Force Use of Synchronous Clear Signals	Forces the Compiler to utilize synchronous clear signals in normal mode logic cells. Enabling this option helps to reduce the total number of logic cells in the design, but can negatively impact the fitting. All the logic cells in a LAB share synchronous control signals.

continued...

Option	Description
HDL Message Level	Specifies the type of HDL messages you want to view, including messages that display processing errors in the HDL source code. **Level1** displays only the most important HDL messages. **Level2** displays most HDL messages, including warning and information based messages. **Level3** displays all HDL messages, including warning and information based messages and alerts about potential design problems or lint errors.
Ignore CARRY Buffers	Ignores CARRY_SUM buffers in the design. The Compiler ignores this option if you apply the option to anything other than an individual CARRY_SUM buffer, or to a design entity containing CARRY_SUM buffers.
Ignore CASCADE Buffers	Ignores CASCADE buffers that are instantiated in the design. The Compiler ignores this option if you apply the option to anything other than an individual CASCADE buffer, or a design entity containing CASCADE buffers.

Table 45. Advanced Synthesis Settings (6 of 13)

Option	Description
Ignore GLOBAL Buffers	Ignores GLOBAL buffers in the design. The Compiler ignores this option if you apply the option to anything other than an individual GLOBAL buffer, or a design entity containing GLOBAL buffers.
Ignore LCELL Buffers	Ignores LCELL buffers in the design. The Compiler ignores this option if you apply the option to anything other than an individual LCELL buffer, or a design entity containing LCELL buffers.
Ignore Maximum Fan-Out Assignments	Directs the Compiler to ignore the Maximum Fan-Out Assignments on a node, an entity, or the whole design.
Ignore ROW GLOBAL Buffers	Ignores ROW GLOBAL buffers in the design. The Compiler ignores this option if you apply the option to anything other than an individual GLOBAL buffer or a design entity containing GLOBAL buffers.
Ignore SOFT Buffers	Ignores SOFT buffers in the design. The Compiler ignores this option if you apply the option to anything other than an individual SOFT buffer or a design entity containing SOFT buffers.

Table 46. Advanced Synthesis Settings (7 of 13)

Option	Description
Ignore translate_off and synthesis_off Directives	Ignores all `translate_off/synthesis_off` synthesis directives in Verilog HDL and VHDL design files. Use this option to disable these synthesis directives and include previously ignored code during elaboration.
Infer RAMs from Raw Logic	Infers RAM from registers and multiplexers. The Compiler initially converts some HDL patterns differing from RAM templates into logic. However, these structures function as RAM. As a result, when you enable this option, the Compiler may substitute the altsyncram IP core instance for them at a later stage. When you enable this assignment, the Compiler may use more device RAM resources and fewer LABs.
Iteration Limit for Constant Verilog Loops	Defines the iteration limit for Verilog loops with loop conditions that evaluate to compile-time constants on each loop iteration. This limit exists primarily to identify potential infinite loops before they exhaust memory or trap the software in an actual infinite loop.
Iteration Limit for non-Constant Verilog Loops	Defines the iteration limit for Verilog HDL loops with loop conditions that do not evaluate to compile-time constants on each loop iteration. This limit exists primarily to identify potential infinite loops before they exhaust memory or trap the software in an actual infinite loop.

Table 47. Advanced Synthesis Settings (8 of 13)

Option	Description
Limit AHDL integers to 32 Bits	Specifies whether an AHDL-based design must have a limit on integer size of 32 bits. The Compiler provides this option for backward compatibility with pre-2000.09 releases of the Quartus software. Such registers do not support integers larger than 32 bits in AHDL.
Maximum DSP Block Usage	Specifies the maximum number of DSP blocks that the DSP block balancer assumes exist in the current device for each partition. This option overrides the usual method of using the maximum number of DSP blocks the current device supports.
Maximum Number of LABs	Specifies the maximum number of LABs that Analysis & Synthesis should try to utilize for a device. This option overrides the usual method of using the maximum number of LABs the current device supports, when the value is non-negative and is less than the maximum number of LABs available on the current device.
Maximum Number of M4K/M9K/M20K/M10K Memory Blocks	Specifies the maximum number of M4K, M9K, M20K, or M10K memory blocks that the Compiler may use for a device. This option overrides the usual method of using the maximum number of M4K, M9K, M20K, or M10K memory blocks the current device supports, when the value is non-negative and is less than the maximum number of M4K, M9K, M20K, or M10K memory blocks available on the current device.

Table 48. Advanced Synthesis Settings (9 of 13)

Option	Description
Maximum Number of Registers Created from Uninferred RAMs	Specifies the maximum number of registers that Analysis & Synthesis uses for conversion of uninferred RAMs. Use this option as a project-wide option or on a specific partition by setting the assignment on the instance name of the partition root. The assignment on a partition overrides the global assignment (if any) for that particular partition. This option prevents synthesis from causing long compilations and running out of memory when many registers are used for uninferred RAMs. Instead of continuing the compilation, the Quartus Prime software issues an error and exits.
NOT Gate Push-Back	Allows the Compiler to push an inversion (that is, a NOT gate) back through a register and implement it on that register's data input if it is necessary to implement the design. When this option is on, a register may power-up to an active-high state, and may need explicit clear during initial operation of the device. The Compiler ignores this option if you apply it to anything other than an individual register or a design entity containing registers. When you apply this option to an output pin that is directly fed by a register, the assignment automatically transfers to that register.
Number of Inverted Registers Reported in Synthesis Report	Specifies the maximum number of inverted registers that the Synthesis report displays.
Number of Protected Registers Reported in Synthesis Report	Specifies the maximum number of protected registers that the Synthesis Report should display.
Number of Removed Registers Reported in Synthesis Migration Checks	Specifies the maximum number of rows that the Synthesis Migration Check report displays.
Number of Swept Nodes Reported in Synthesis Report	Specifies the maximum number of swept nodes that the Synthesis Report displays. A swept node is any node which was eliminated from your design because the Compiler found the node to be unnecessary.
Number of Rows Reported in Synthesis Report	Specifies the maximum number of rows that the Synthesis report displays.
Optimization Technique	Specifies an overall optimization goal for Analysis & Synthesis. The Compiler can maximize synthesis processing for performance, minimize logic usage, or balance high performance with minimal logic usage.

Table 49. Advanced Synthesis Settings (10 of 13)

Option	Description
Perform WYSIWYG Primitive Resynthesis	Specifies whether to perform WYSIWYG primitive resynthesis during synthesis. This option uses the setting specified in the **Optimization Technique** logic option.
Power-Up Don't Care	Causes registers that do not have a **Power-Up Level logic** option setting to power-up with a don't care logic level (X). When the **Power-Up Don't Care** option is on, the Compiler determines when it is beneficial to change the power-up level of a register to minimize the area of the design. The Compiler maintains a power-up state of zero, unless there is an immediate area advantage.
Power Optimization During Synthesis	Controls the power-driven compilation setting of Analysis & Synthesis. This option determines how aggressively Analysis & Synthesis optimizes the design for power. When this option is **Off**, the Compiler does not perform any power optimizations. **Normal compilation** performs power optimizations as long as they are not expected to reduce design performance. **Extra effort** performs additional power optimizations which may reduce design performance.

Table 50. Advanced Synthesis Settings (11 of 13)

Option	Description
Remove Duplicate Registers	Removes a register if it is identical to another register. If two registers generate the same logic, the Compiler deletes the duplicate. The first instance fans-out to the duplicates destinations. Also, if the deleted register contains different logic option assignments, the Compiler ignores the options. This option is useful if you wish to prevent the Compiler from removing intentionally duplicate registers. The Compiler ignores this option if you apply it to anything other than an individual register or a design entity containing registers.
Remove Redundant Logic Cells	Removes redundant LCELL primitives or WYSIWYG primitives. Turning this option on optimizes a circuit for area and speed. The Compiler ignores this option if you apply it to anything other than a design entity.
Report Connectivity Checks	Specifies whether the Synthesis report includes the panels in the Connectivity Checks folder.
Report Parameter Settings	Specifies whether the Synthesis report includes the panels in the Parameter Settings by Entity Instance folder.
Report Source Assignments	Specifies whether the Synthesis report includes the panels in the Source Assignments folder.

Table 51. Advanced Synthesis Settings (12 of 13)

Option	Description
Resource Aware Inference for Block RAM	Specifies whether RAM, ROM, and shift-register inference should take the design and device resources into account.
Restructure Multiplexers	Reduces the number of logic elements required to implement multiplexers in a design. This option is useful if your design contains buses of fragmented multiplexers. This option repacks multiplexers more efficiently for area, allowing the design to implement multiplexers with a reduced number of logic elements: • **On**—minimizes your design area, but may negatively affect design clock speed (f_{MAX}). • **Off**—disables multiplexer restructuring; it does not decrease logic element usage and does not affect design clock speed (f_{MAX}). • **Auto**—allows the Quartus Prime software to determine whether multiplexer restructuring should be enabled. The **Auto** setting decreases logic element usage, but may negatively affect design clock speed (f_{MAX}).

continued...

Option	Description
SDC Constraint Protection	Verifies `.sdc` constraints in register merging. This option helps to maintain the validity of `.sdc` constraints through compilation.
Safe State Machine	Directs the Compiler to implement state machines that can recover from an illegal state.
Shift Register Replacement – Allow Asynchronous Clear Signal	Allows the Compiler to find a group of shift registers of the same length that can be replaced with the altshift_taps IP core. The shift registers must all use the same `aclr` signals, must not have any other secondary signals, and must have equally spaced taps that are at least three registers apart. To use this option, you must turn on the **Auto Shift Register Replacement** logic option.

Table 52. Advanced Synthesis Settings (13 of 13)

Option	Description
State Machine Processing	Specifies the processing style used to compile a state machine. You can use your own **User-Encoded** style, or select **One-Hot**, **Minimal Bits**, **Gray**, **Johnson**, **Sequential,** or **Auto** (Compiler-selected) encoding.
Strict RAM Replacement	When this option is **On**, the Compiler replace RAM only if the hardware matches the design exactly.
Synchronization Register Chain Length	Specifies the maximum number of registers in a row that the Compiler considers as a synchronization chain. Synchronization chains are sequences of registers with the same clock and no fan-out in between, such that the first register is fed by a pin, or by logic in another clock domain. The Compiler considers these registers for metastability analysis. The Compiler prevents optimizations of these registers, such as retiming. When gate-level retiming is enabled, the Compiler does not remove these registers. The default length is set to two.
Synthesis Effort	Controls the synthesis trade-off between compilation speed, performance, and area. The default is **Auto**. You can select **Fast** for faster compilation speed at the cost of performance and area.
Timing-Driven Synthesis	Allows synthesis to use timing information to better optimize the design. The **Timing-Driven Synthesis** logic option impacts the following **Optimization Technique** options: • **Optimization Technique Speed**—optimizes timing-critical portions of your design for performance at the cost of increasing area (logic and register utilization) • **Optimization Technique Balanced**—also optimizes the timing-critical portions of your design for performance, but the option allows only limited area increase • **Optimization Technique Area**—optimizes your design only for area

6.9 Fitter Settings Reference

Use Fitter settings to customize the place and route of your design. Click **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Fitter)** to access Fitter settings.

Table 53. Advanced Fitter Settings (1 of 8)

Option	Description
ALM Register Packing Effort	Guides aggressiveness of the Fitter in packing ALMs during register placement. Use this option to increase secondary register locations. Increasing ALM packing density may lower the number of ALMs needed to fit the design, but it may also reduce routing flexibility and timing performance.

continued...

Option	Description
	• **Low**—the Fitter avoids ALM packing configurations that combine LUTs and registers which have no direct connectivity. Avoiding these configurations may improve timing performance but increases the number of ALMs to implement the design. • **Medium**—the Fitter allows some configurations that combine unconnected LUTs and registers to be implemented in ALM locations. The Fitter makes more use of secondary register locations within the ALM. • **High**—the Fitter enables all legal and desired ALM packing configurations. In dense designs, the Fitter automatically increases the ALM register packing effort as required to enable the design to fit.
Allow Register Duplication	Allows the Compiler to duplicate registers to improve design performance. When you enable this option, the Compiler copies registers and moves some fan-out to this new node. This optimization improves routability and can reduce the total routing wire in nets with many fan-outs. If you disable this option, this disables optimizations that retime registers.
Allow Register Merging	Allows the Compiler to remove registers that are identical to other registers in the design. When you enable this option, in cases where two registers generate the same logic, the Compiler deletes one register, and the remaining registers fan-out to the deleted register's destinations. This option is useful if you wish to prevent the Compiler from removing intentional use of duplicate registers. If you disable register merging, the Compiler disables optimizations that retime registers.
Allow Delay Chains	Allows the Fitter to choose the optimal delay chain to meet t_{SU} and t_{CO} timing requirements for all I/O elements. Enabling this option may reduce the number of t_{SU} violations, while introducing a minimal number of t_H violations. Enabling this option does not override delay chain settings on individual nodes.
Auto Delay Chains for High Fanout Input Pins	Allows the Fitter to choose how to optimize the delay chains for high fan-out input pins. You must enable **Auto Delay Chains** to enable this option. Enabling this option may reduce the number of t_{SU} violations, but the compile time increases significantly, as the Fitter tries to optimize the settings for all fan-outs.
Auto Fit Effort Desired Slack Margin	Specifies the default worst-case slack margin the Fitter maintains for. If the design is likely to have at least this much slack on every path, the Fitter reduces optimization effort to reduce compilation time.

Table 54. Advanced Fitter Settings (2 of 8)

Option	Description
Auto Global Clock	Allows the Compiler to choose the global clock signal. The Compiler chooses the signal that feeds the most clock inputs to flip-flops. This signal is available throughout the device on the global routing paths. To prevent the Compiler from automatically selecting a particular signal as global clock, set the **Global Signal** option to **Off** on that signal.
Auto Global Register Control Signals	Allows the Compiler to choose global register control signals. The Compiler chooses signals that feed the most control signal inputs to flip-flops (excluding clock signals) as the global signals. These global signals are available throughout the device on the global routing paths. Depending on the target device family, these control signals can include asynchronous clear and load, synchronous clear and load, clock enable, and preset signals. If you want to prevent the Compiler from automatically selecting a particular signal as a global register control signal, set the **Global Signal** option to **Off** on that signal.
Auto Packed Registers	Allows the Compiler to combine a register and a combinational function, or to implement registers using I/O cells, RAM blocks, or DSP blocks instead of logic cells. This option controls how aggressively the Fitter combines registers with other function blocks to reduce the area of the design. Generally, the **Auto** or **Sparse Auto** settings are appropriate.

continued...

Option	Description
	The other settings limit the flexibility of the Fitter to combine registers with other function blocks and can result in no fits. • **Auto**—the Fitter attempts to achieve the best performance with good area. If necessary, the Fitter combines additional logic to reduce the area of the design to within the current device. • **Sparse Auto**—the Fitter attempts to achieve the highest performance, but may increase device usage without exceeding the device logic capacity. • **Off**—the Fitter does not combine registers with other functions. The **Off** setting severely increases the area of the design and may cause a no fit. • **Sparse**—the Fitter combines functions in a way which improves performance for many designs. • **Normal**—the Fitter combines functions that are expected to maximize design performance and reduce area. • **Minimize Area**—the Fitter aggressively combines unrelated functions to reduce the area required for placing the design, at the expense of performance. • **Minimize Area with Chains**—the Fitter even more aggressively combines functions that are part of register cascade chains or can be converted to register cascade chains. If this option is set to any value but **Off**, registers combine with I/O cells to improve I/O timing. This remains true as long as the **Optimize IOC Register Placement For Timing** option is enabled.
Auto RAM to MLAB Conversion	Specifies whether the Fitter converts RAMs of **Auto** block type to use LAB locations. If this option is set to **Off,** only MLAB cells or RAM cells with a block type setting of **MLAB** use LAB locations to implement memory.
Auto Register Duplication	Allows the Fitter to automatically duplicate registers within a LAB that contains empty logic cells. This option does not alter the functionality of the design. The Compiler ignores the **Auto Register Duplication** option if you select **OFF** as the setting for the **Logic Cell Insertion -- Logic Duplication** logic option. Turning on this option allows the **Logic Cell Insertion -- Logic Duplication** logic option to improve a design's routability, but can make formal verification of a design more difficult.

Table 55. Advanced Fitter Settings (3 of 8)

Option	Description
Enable Bus-Hold Circuitry	Enables bus-hold circuitry during device operation. When this option is on, a pin retains its last logic level when it is not driven, and does not go to a high impedance logic level. Do not use this option at the same time as the **Weak Pull-Up Resistor** option. The Compiler ignores this option if you apply it to anything other than a pin.
Equivalent RAM and MLAB Paused Read Capabilities	Specifies whether RAMs implemented in MLAB cells must have equivalent paused read capabilities as RAMs implemented in block RAM. Pausing a read is the ability to keep around the last read value when reading is disabled. Allowing differences in paused read capabilities provides the Fitter more flexibility in implementing RAMs using MLAB cells. To allow the Fitter the most flexibility in deciding which RAMs are implemented using MLAB cells, set this option to **Don't Care**. The following options are available: • **Don't Care**—the Fitter can convert RAMs to MLAB cells, even if they do not have equivalent paused read capabilities to a block RAM implementation. The Fitter generates an information message about RAMs with different paused read capabilities. • **Care**—the Fitter does not convert RAMs to MLAB cells unless they have the equivalent paused read capabilities to a block RAM implementation.
Equivalent RAM and MLAB Power Up	Specifies whether RAMs implemented in MLAB cells must have equivalent power-up conditions as RAMs implemented in block RAM. Power-up conditions occur when the device powers-up or globally resets. Allowing non-equivalent power-up conditions provides the Fitter more flexibility in implementing RAMs using MLAB cells.

continued...

Option	Description
	To allow the Fitter the most flexibility in deciding which RAMs are implemented using MLAB cells, set this option to **Auto** or **Don't Care**. The following options are available: • **Auto**—the Fitter may convert RAMs to MLAB cells, even if the MLAB cells lack equivalent power-up conditions to a block RAM implementation. The Fitter also outputs a warning message about RAMs with non-equivalent power up conditions. • **Don't Care**—the same behavior as **Auto** applies, but the message is an information message. • **Care**—the Fitter does not convert RAMs to MLAB cells unless they have equivalent power up conditions to a block RAM implementation.
Final Placement Optimizations	Specifies whether the Fitter performs final placement optimizations. Performing final placement optimizations may improve timing and routability, but may also require longer compilation time.
Fitter Aggressive Routability Optimizations	Specifies whether the Fitter aggressively optimizes for routability. Performing aggressive routability optimizations may decrease design speed, but may also reduce routing wire usage and routing time. The **Automatically** setting allows the Fitter to decide whether aggressive routability is beneficial.

Table 56. Advanced Fitter Settings (4 of 8)

Option	Description
Fitter Effort	Specifies the level of physical synthesis optimization during fitting: • **Auto**—adjusts the Fitter optimization effort to minimize compilation time, while still achieving the design timing requirements. Use the **Auto Fit Effort Desired Slack Margin** option to apply sufficient optimization effort to achieve additional timing margin. • **Standard**—uses maximum effort regardless of the design's requirements, leading to higher compilation time and more margin on easier designs. For difficult designs, **Auto** and **Standard** both use maximum effort.
Fitter Initial Placement Seed	Specifies the seed for the current design. The value can be any non-negative integer value. By default, the Fitter uses a seed of 1. The Fitter uses the seed as the initial placement configuration when optimizing design placement to meet timing requirements f_{MAX}. Because each different seed value results in a somewhat different fit, you can try several different seeds to attempt to obtain superior fitting results. The seeds that lead to the best fits for a design may change if the design changes. Also, changing the seed may or may not result in a better fit. Therefore, specify a seed only if the Fitter is not meeting timing requirements by a small amount. *Note:* You can also use the Design Space Explorer II (DSEII) to sweep complex flow parameters, including the seed, in the Quartus Prime software to optimize design performance.
Logic Cell Insertion	Allows the Fitter to automatically insert buffer logic cells between two nodes without altering the functionality of the design. The Compiler creates buffer logic cells from unused logic cells in the device. This option also allows the Fitter to duplicate a logic cell within a LAB when there are unused logic cells available in a LAB. Using this option can increase compilation time. The default setting of **Auto** allows these operations to run when the design requires them to fit the design.
MLAB Add Timing Constraints for Mixed-Port Feed-Through Mode Setting Don't Care	Specifies whether the TimeQuest Timing Analyzer evaluates timing constraints between the write and the read operations of the MLAB memory block. Performing a write and read operation simultaneously at the same address might result in metastability issues because no timing constraints between those operations exist by default. Turning on this option introduces timing constraints between the write and read operations on the MLAB memory block and thereby avoids metastability issues. However, turning on this option degrades the performance of the MLAB memory blocks. If your design does not perform write and read operations simultaneously at the same address, you do not need to set this option.

Table 57. Advanced Fitter Settings (5 of 8)

Option	Description
Optimize Design for Metastability	This setting improves the reliability of the design by increasing its Mean Time Between Failures (MTBF). When you enable this setting, the Fitter increases the output setup slacks of synchronizer registers in the design. This slack can exponentially increase the design MTBF. This option only applies when using the TimeQuest Timing Analyzer for timing-driven compilation. Use the TimeQuest Timing Analyzer `report_metastability` command to review the synchronizers detected in your design and to produce MTBF estimates.
Optimize Hold Timing	Directs the Fitter to optimize hold time within a device to meet timing requirements and assignments. The following settings are available: • **I/O Paths and Minimum TPD Paths—**directs the Fitter to meet the following timing requirements and assignments: — t_H from I/O pins to registers. — Minimum t_{CO} from registers to I/O pins. — Minimum t_{PD} from I/O pins or registers to I/O pins or registers. • **All Paths**—directs the Fitter to meet the following timing requirements and assignments: — t_H from I/O pins to registers. — Minimum t_{CO} from registers to I/O pins. — Minimum t_{PD} from I/O pins or registers to I/O pins or registers. When you disable the **Optimize Timing** logic option, the **Optimize Hold Timing** option is not available.
Optimize IOC Register Placement for Timing	Specifies whether the Fitter optimizes I/O pin timing by automatically packing registers into I/Os to minimize delays. • **Normal**—the Fitter opportunistically packs registers into I/Os that should improve I/O timing. • **Pack All I/O Registers**— the Fitter aggressively packs any registers connected to input, output, or output enable pins into I/Os, unless prevented by user constraints or other legality restrictions. • **Off**—performs no periphery to core optimization.
Optimize Multi-Corner Timing	Directs the Fitter to consider all timing corners during optimization to meet timing requirements. These timing delay corners include both fast-corner timing and slow-corner timing. By default, this option is **On**, and the Fitter optimizes designs considering multi-corner delays in addition to slow-corner delays. When this option is **Off**, the Fitter optimizes designs considering only slow-corner delays from the slow-corner timing model (slowest manufactured device for a given speed grade, operating in low-voltage conditions). Turning this option **On** typically creates a more robust design implementation across process, temperature, and voltage variations. When you turn **Off** the **Optimize Timing** option, the **Optimize Multi-Corner Timing** option is not available.
Optimize Timing	Specifies whether the Fitter optimizes to meet the maximum delay timing requirements (for example, clock cycle time). By default, this option is set to **Normal compilation**. Turning this option **Off** helps fit designs that with extremely high interconnect requirements. Turning this option **Off** can also reduce compilation time at the expense of timing performance (because the Fitter ignores the design's timing requirements). If this option is **Off**, other Fitter timing optimization options have no effect (such as **Optimize Hold Timing**).

Table 58. Advanced Fitter Settings (6 of 8)

Option	Description
Optimize Timing for ECOs	Controls whether the Fitter optimizes to meet the user's maximum delay timing requirements (for example, clock cycle time, t_{SU}, t_{CO}) during ECO compiles. By default, this option is set to **Off**. Turning it **On** can improve timing performance at the cost of compilation time.
Perform Clocking Topology Analysis During Routing	Directs the Fitter to perform an analysis of the design's clocking topology and adjust the optimization approach on paths with significant clock skew. Enabling this option may improve hold timing at the cost of increased compile time.

continued...

Option	Description
Periphery to Core Placement and Routing Optimization	Specifies whether the Fitter should perform targeted placement and routing optimization on direct connections between periphery logic and registers in the FPGA core. The following options are available: • **Auto**—the Fitter automatically identifies transfers with tight timing windows, places the core registers, and routes all connections to or from the periphery. The Fitter performs these placement and routing decisions before the rest of core placement and routing. This sequence ensures that these timing-critical connections meet timing, and also avoids routing congestion. • **On**— the Fitter optimizes all transfers between the periphery and core registers, regardless of timing requirements. Do not set this option to **On** globally. Instead, use the Assignment Editor to assign optimization to a targeted set of nodes or entities. • **Off**—the Fitter performs no periphery to core optimization.
Physical Synthesis	Increases circuit performance by performing combinational and sequential optimization during fitting.
Placement Effort Multiplier	Specifies the relative time the Fitter spends in placement. The default value is 1.0 and legal values must be greater than 0. Specifying a floating-point number allows you to control the placement effort. A higher value increases CPU time but may improve placement quality. For example, a value of '4' increases fitting time by approximately 2 to 4 times but may increase quality.
Power Optimization During Fitting	Directs the Fitter to perform optimizations targeted at reducing the total power devices consume. The available settings for power-optimized fitting are: • **Off**—performs no power optimizations. • **Normal compilation**—performs power optimizations that are unlikely to adversely affect compilation time or design performance. • **Extra effort**—performs additional power optimizations that might affect design performance or result in longer compilation time.

Table 59. Advanced Fitter Settings (7 of 8)

Option	Description
Programmable Power Maximum High-Speed Fraction of Used LAB Tiles	Sets the upper limit on the fraction of the high-speed LAB tiles. Legal values must be between 0.0 and 1.0. The default value is 1.0. A value of 1.0 means that there is no restriction on the number of high-speed tiles, and the Fitter uses the minimum number needed to meet the timing requirements of your design. Specifying a value lower than 1.0 might degrade timing quality, because some timing critical resources might be forced into low-power mode.
Programmable Power Technology Optimization	Controls how the Fitter configures tiles to operate in high-speed mode or low-power mode. The following options are available: • **Automatic**—specifies that the Fitter minimizes power without sacrificing timing performance. • **Minimize Power Only**—specifies that the Fitter sets the maximum number of tiles to operate in low-power mode. • **Force All Used Tiles to High Speed**—specifies that the Fitter sets all used tiles to operate in high-speed mode. • **Force All Tiles with Failing Timing Paths to High Speed**—sets all failing paths to high-speed mode. For designs that meet timing, the behavior of this setting is similar to the **Automatic** setting. For designs that fail timing, all paths with negative slack are put in high-speed mode. This mode likely does not increase the speed of the design, and it may increase static power consumption. This mode may assist in determining which logic paths need to be re-designed to close timing.

continued...

Option	Description
Regenerate Full Fit Reports During ECO Compiles	Controls whether the Fitter report is regenerated during ECO compilation. By default, this option is set to **Off**. Turning it **On** regenerates the report at the cost of compilation time.
Router Timing Optimization Level	Controls how aggressively the router tries to meet timing requirements. Setting this option to **Maximum** can increase design speed slightly, at the cost of increased compile time. Setting this option to **Minimum** can reduce compile time, at the cost of slightly reduced design speed. The default value is **Normal**.
Run Early Place during compilation	Enables the Early Place Fitter stage during full compilation. Turning on this setting may increase Fitter processing time.

Table 60. Advanced Fitter Settings (8 of 8)

Option	Description
Synchronizer Identification	Specifies how the Compiler identifies synchronization register chain registers for metastability analysis. A synchronization register chain is a sequence of registers with the same clock with no fan-out in between, which is driven by a pin or logic from another clock domain. The following options are available: • **Off**—the TimeQuest Timing Analyzer does not identify the specified registers, or the registers within the specified entity, as synchronization registers. • **Auto**—the TimeQuest Timing Analyzer identifies valid synchronization registers that are part of a chain with more than one register that contains no combinational logic. Use the **Auto** setting to generate a report of possible synchronization chains in your design. • **Forced if Asynchronous**—the TimeQuest Timing Analyzer identifies synchronization register chains if the software detects an asynchronous signal transfer, even if there is combinational logic or only one register in the chain. • **Forced**—the TimeQuest Timing Analyzer identifies the specified register, or all registers within the specified entity, as synchronizers. Only apply the **Forced** option to the entire design. Otherwise, all registers in the design identify as synchronizers. The Fitter optimizes the registers that it identifies as synchronizers for improved Mean Time Between Failure (MTBF), as long as you enable **Optimize Design for Metastability**. If a synchronization register chain is identified with the **Forced** or **Forced if Asynchronous** option, then the TimeQuest Timing Analyzer reports the metastability MTBF for the chain when it meets the design timing requirements.
Treat Bidirectional Pin as Output Pin	Specifies that the Fitter treats the bidirectional pin as an output pin, meaning that the input path feeds back from the output path.
Weak Pull-Up Resistor	Enables the weak pull-up resistor when the device is operating in user mode. This option pulls a high-impedance bus signal to VCC. Do not enable this option simultaneously with the **Enable Bus-Hold Circuitry** option. The Fitter ignores this option if you apply to anything other than a pin.

6.10 Document Revision History

This document has the following revision history.

Table 61. Document Revision History

Date	Version	Changes
2017.05.08	17.0.0	• Added reference to initial compilation support for Cyclone 10 GX devices. • Described concurrent analysis following Early Place. • Updated Compilation Dashboard images for TimeQuest, Report, Setting, and Concurrent Analysis controls. • Updated description for Auto DSP Block Replacement in Advanced Synthesis Settings. • Updated Advanced Fitter Settings for Allow Register Retiming, and for removal of obsolete SSN Optimization option.

continued...

Date	Version	Changes
		• Added Prevent Register Retiming topic. • Added Preserve Registers During Synthesis topic. • Removed limitation for **Safe State Machine** logic option. • Added references to Partial Reconfiguration and Block-Based Design Flows.
2016.10.31	16.1.0	• Implemented Intel re-branding. • Described Compiler snapshots and added Analyzing Snapshot Timing topic. • Updated project directory structure diagram. • Described new Fitter stage menu commands and reports. • Added description of Early Place Flow, Implement Flow, and Finalize Flow. • Added description of Incremental Optimization in the Fitter. • Reorganized order of topics in chapter. • Removed deprecated **Per-Stage Compilation (Beta)** Compilation Flow.
2016.05.03	16.0.0	• Added description of Fitter Plan, Place and Route stages, reporting, and optimization. • Added **Per-Stage Compilation (Beta)** Compilation Flow • Added Compilation Dashboard information. • Removed support for **Safe State Machine** logic option. Encode safe states in RTL. • Added Generating Dynamic Synthesis Reports topic. • Updated Quartus project directory structure.
2015.11.02	15.1.0	• First version of document.

Related Links

Altera Documentation Archive

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

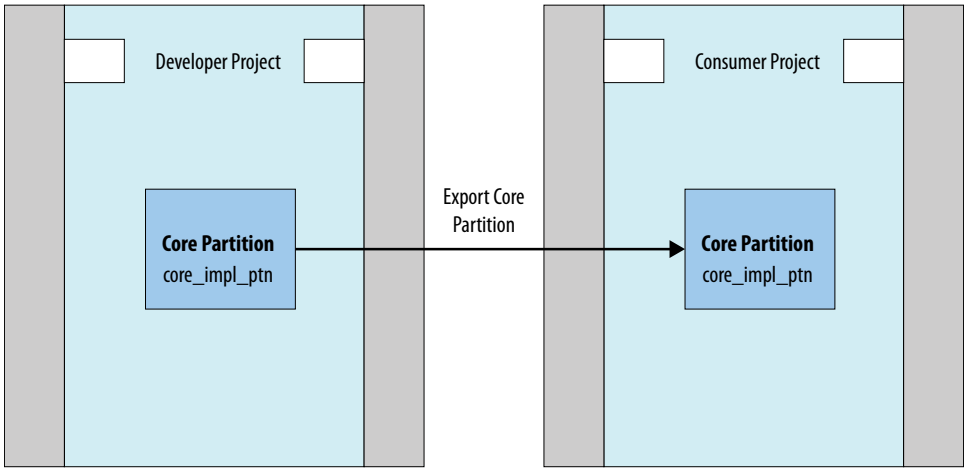
7 Block-Based Design Flows

The Intel Quartus Prime Pro Edition software supports preservation and reuse of design blocks in one or more projects. You can reuse synthesized, placed, or routed 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 your project, and then empty, preserve, or export the contents of that design partition after compilation. The 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 at source, synthesis, or final compilation stage.
- **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.

Figure 71. Design Block Reuse of Core Partition



This chapter first describes using incremental block-based compilation to create and optimize a core design partition within a project. *Design Block Reuse* then describes exporting the core and periphery partitions for reuse in another project.

7.1 Block-Based Design Terms

This table defines the block-based design terms in this document:

Table 62. Block-Based Design Terms

Term	Description
Black Box File	In core partition reuse, a stub file that contains only port definitions, without logic. Include parameters passed to the module to ensure that the configuration matches the implementation in the consumer project. The file format can be any RTL source.
Block	A design partition that you preserve, empty, or export.
Consumer Project	A Quartus Prime design project that consumes the design partition exported from a developer project.
Core Resources	FPGA resources for the implementation of core logic, such as LUTs, flipflops, M20K memory blocks, DSPs, and I/O PLLs. A partition of core resources cannot include periphery resources.
Design Partition	A logical, named, hierarchical boundary that you can assign to a design entity. The Compiler preserves the results of each compilation stage for each partition independently. Export and import a design partition to reuse a design block in another project.
Developer Project	A Quartus Prime project in which you develop a reusable design partition for export.
Floorplanning	Planning the physical layout of FPGA device resources. Creating a design floorplan, or floorplanning, is the manual process of mapping the logical design hierarchy and periphery to physical regions in the device and I/O.
LogicLock Plus Region Constraints	Constrains the placement and routing of logic to a specific region in the target device. You can specify the region origin, height, and width, along with any of the following options: • **Reserved**—prevents the Fitter from placing non-member logic within the region. • **Core-Only**—applies the constraint only to core logic in the region, and does not include periphery logic in the region. • **Routing Region**—restricts the routing of the members in the region to that area. The routing region is non-exclusive and other resources can use that routing area.
Periphery Resources	FPGA resources for the implementation of device periphery elements, such as I/O, HSSIO, memory interfaces, and PCIe*.
Project	The Quartus Prime software organizes the RTL, settings, constraints, and revisions of your design within a project. When you define the project in the Quartus Prime GUI, the Quartus Prime Project File (`.qpf`) stores the project name and references each project revision that you create.
Project Revision	A named collection of settings and constraints for one version of your Quartus Prime project. A Quartus Prime settings File (`.qsf`) preserves the revisions of your project. The Quartus Prime project can contain multiple revisions. Revisions allows you to organize several versions of your design within a single project.
root_partition	The Quartus Prime software automatically creates a top-level root_partition for the entire project. You can export a periphery partition as the root_partition when reusing periphery blocks in another project.
Snapshot	The Quartus Prime Compiler generates a snapshot of the compilation database after each stage. You can optionally preserve or export a snapshot.

Table 63. Block-Based Design Flows Comparison

This table summarizes the differences between incremental block-based compilation and design block reuse flows.

		Incremental Block-Based Compilation	Design Block Reuse
Reuse Context		Reuses design block in the same project	Reuses design block in a different project
Snapshot Preservation		None, Source, Synthesis, Final	Preserves the Synthesis, Planned, Early Placed, Placed, Routed, Final snapshot as a Partition Database File (`.qdb`)
Empty Partitions		Yes	No
Core Partitions		Yes	Yes
Root Partitions		No	Yes
Requires Additional Files		None	• Partition database `.qdb` • Black box port definitions file (core partition only) • Timing constraints `.sdc`
Requires Floorplanning		Not required	Required for periphery partition reuse
Resources	Periphery Partition	Not available	Periphery or core resources
	Core Partition	Core resources only	Core resources only

7.2 Incremental Block-Based Compilation

In a block-based compilation flow, you can split a large design into partitions.

Block-based compilation preserves placement and routing results and performance of unchanged partitions in the design. This can reduce design iteration time by focusing new compilations on changed design partitions only. Block-based compilation includes preserved partitions with any changes. Additionally, you can target optimization techniques to specific design partitions, while leaving other partitions unchanged. You can also use empty partitions to indicate that parts of your design are incomplete or missing, while you compile the rest of your design.

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

When you plan your design code and hierarchy, ensure that each design entity is created in a separate file. Entities can then remain independent when you make source code changes in the file. The netlists act as source files for block-based compilation.

7.2.1 Block-Based Design Models

You can plan block-based design based on your design requirements. The following models are typical examples.

Top-Down Design Model

Empty partitions allow you to use a top-down design approach without all the modules. Create a project and partitions with or without the existing RTL. Marking partitions as empty allows the tools to quickly elaborate the design without synthesizing everything. Complete and verify each partition before integration into the design. Unnecessary partitions can impact performance. If required, partitions can be removed as the design takes shape to allow the tools to optimize the design further.

If you use empty partitions, the Compiler removes any existing synthesis, placement, and routing information for that partition. If you remove the empty option from that partition, the Compiler reimplements the partition from the source and preserves nothing from previous runs. To use the empty partition feature, set the preservation level to source. If you set the preservation level to Synthesis or Final, the Compiler ignores the emtpy option.

Bottom-Up Design Model

In a highly utilized device, a bottom-up design can be effective to meet your design requirements. In this model, you implement the design in partitions, analyze the results, and preserve the critical partitions. When using the bottom-up model, do not preserve everything, because the locked resources may remove solutions for other parts of the design. Best practice indicates that you only preserve critical areas.

Periphery Planning Design Model

Use empty partitions to quickly plan the periphery. Since the periphery only depends on periphery resources, you can mark core partitions as empty. This model decreases the need for the complete RTL initially, and reduces the initial compile times for large designs. Use this model in conjunction with physical periphery planning using Pin Planner, Chip Planner, or BluePrint design planner.

7.2.2 Block-Based Design Partition Planning

To use incremental block-based compilation, you must first create design partitions from hierarchical instances in your design. Planning your partitions in advance prevents having to re-partition mid-process, and limits the likelihood of partition over use.

Partitions facilitate incremental block-based compilation by allowing separate synthesis, placement, and routing of each partition, and by preventing Compiler optimizations across partition boundaries. To identify your design's hierarchy, you must first run **Analysis & Elaboration**, or run any compilation flow that includes this step. When you create a design partition, the Quartus Prime software automatically generates a partition name based on the instance name and hierarchy path. All design partition names must be unique in the design and can consist only of alphanumeric name characters and underscore (_) characters.

You can create design partitions from the **Hierarchy** tab in the Project Navigator, or the **Design Partitions** window:

- In the Project Navigator, right-click an instance in the **Hierarchy** tab, then click **Design Partition ➤ Set as Design Partition**.
- Press Alt-D to open the **Design Partitions Window**, then double-click **<<new>>** to create a partition.
- Click **Assignments ➤ Design Partitions Window ➤** then double-click **<<new>>** to create a partition.

Note: You can create a partition at any stage after elaboration, but creating your partitions early in the design cycle allows the software to optimize your design effectively.

For example, if two entities each contain an inverter on the input and output respectively, normally they cancel out logically. You can reduce that circuit to a single net without using `not` elements. This type of reduction reduces the cell delay, interconnect delay, and increases the f_{MAX}. Creating a partition boundary limits the Compiler's ability to merge partition logic with other parts of the design.

Creating or removing a partition on previously unpartitioned modules may change the synthesis and subsequently the implementation. Delayed partitioning can affect verification efforts, because of the changes to physical implementation created by the new partition. While creating partitions late in the design cycle, you must follow these rules:

1. Remove all periphery resources from the entity.
2. Tunnel all periphery resource ports to the top level.
3. Implement the periphery resource in the root partition.

Related Links

- BluePrint Design Planning
- Design Floorplan Analysis in the Chip Planner

7.2.3 Design Partition Guidelines

Block-based design flows require the use of design partitions to define the logical boundaries of design blocks for reuse. You can define a design partition that contains various FPGA core resources, including LUTs, flipflops, M20K memory blocks, DSPs, and PLLs. A partition can also contain periphery resources, such as I/O, HSSIO, EMIF, and PCIe periphery elements. Clock routing resources belong to the root partition but are not preserved with partitions.

Every Quartus Prime project includes a single root partition. The root partition contains all the periphery resources and can also include core resources. Each project can include several core partitions.

Follow these guidelines when using design partitions for block-based design flows:

- Each core partition can only contain core resources.
- Core partitions cannot contain any periphery resources, like HSSIO.
- Once exported, you cannot modify a partition in any way that effects the partition's compilation snapshot.
- You cannot set a partition that references a `.qdb` file (an imported partition) to preserved or empty.
- You cannot define placement constraints for an imported Final snapshot, because the placement and routing is already complete and defined in the snapshot `.qdb`.
- You can define placement constraints for a Synthesized snapshot, because the Synthesized snapshot does not include place and route data.

Table 64. Core and Periphery FPGA Resources

Resource	Core	Periphery
ALM	Yes	Yes
ATX PLL (Transceiver)	No	Yes
CMU PLL (Transceiver)	No	Yes
Combinational ALUTs	Yes	Yes
Configuration	No	Yes
Dedicated Logic Registers	Yes	Yes
DSP	Yes	Yes
fPLL (Transceiver)	No	Yes
HSSIO and components	No	Yes
IO Register	No	Yes
IO Pads	No	Yes
IO PLLs	Yes	Yes
LAB	Yes	Yes
LUT	Yes	Yes
M20K	Yes	Yes
MLAB	Yes	Yes
PCIe Hard IPs	No	Yes
Secondary logic registers	Yes	Yes

Partition Planning Recommendations

- Plan for partitions, but only implement as needed. Excessive partitioning can impact performance.
- Use top-down design methods. Plan the hierarchy at a high level, then work down as required.
- Plan for design reuse and/or additional functionality when planning the hierarchy and periphery.
- Plan the periphery to help segregate and implement periphery resources in the root partition. Assigning periphery resources to core modules prevents creation of a partition. Proper placement of periphery resources also enables the Quartus Prime software to achieve better placement.
- Plan and create centralized clock and reset modules. This technique avoids periphery and core resource conflicts, and avoids hidden clocks.
- Group modules that can logically share a partition. This technique creates less partition boundary ports and allows maximum optimization within the partition.
- Create the smallest partition that fully contains your module. A large partition can absorb resources that are better used for some other part of the design.
- Register module boundaries and use synchronous design practices. Register boundaries make good partition boundaries. Combinational elements on the module boundaries can affect optimization when partitioned, and are not optimal.
- Avoid late cycle efforts to force a partition. This can be a complex task and requires additional structural changes, verification time, and documentation efforts.
- Avoid grouping unrelated logic into a single partition. If left as separate modules, the tools can optimize smaller modules easier.
- Avoid grouping duplicate modules(lanes) to the same partition, unless they share common parent logic. If partitions are required, create a partition for each lane and let the tools optimize each lane individually. Smaller partitions are easier to optimize and implement.
- When using hierarchically related partitions, the child partition must have a higher preservation level than the parent. If the parent partition has a higher preservation level the child's, the Compiler ignores the preservation level. You define preservation level with the `preserve` attribute for the partition, or as part of data in the `.qdb` file.

7.2.4 PLL Partition Guidelines

You can define and reuse design partitions that include PLLs. Root partitions can contain any type of PLL. However, core partitions can only contain I/O PLLs. The Fitter returns an error if a core partition contains a transceiver PLL. To correct the error, you must remove the transceiver PLL from the core partition.

Carefully plan the clocking scheme to account for the I/O PLLs and avoid generating clocks from the core partitions. Creating a centralized clocking module included at the top has the following advantages:

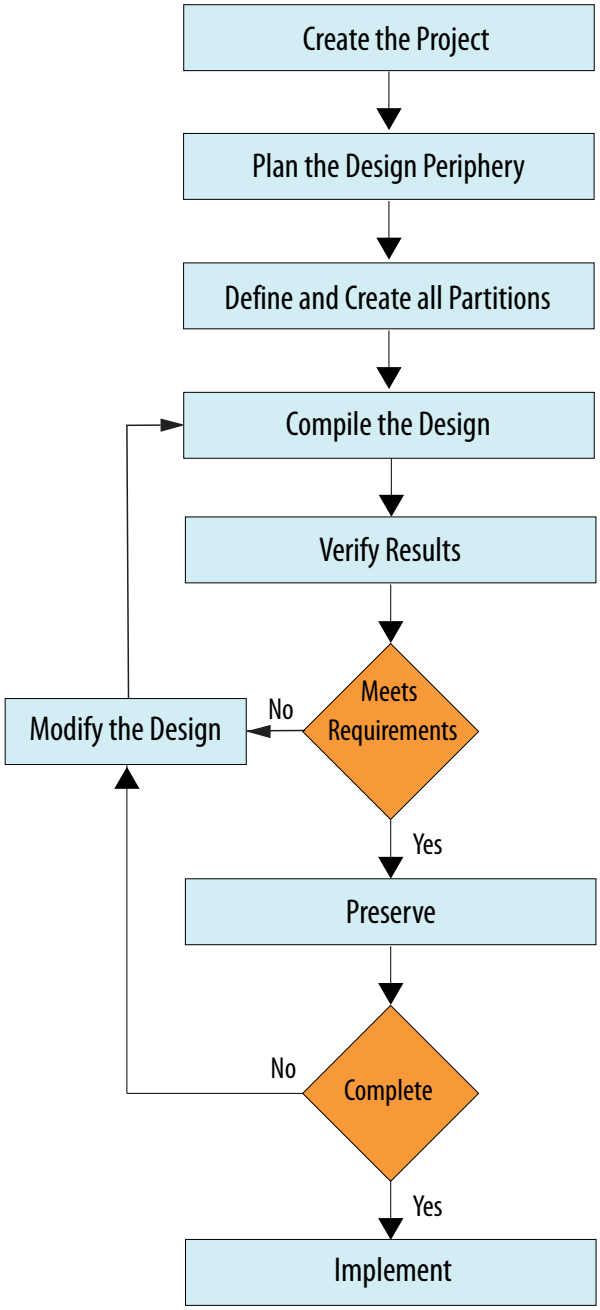
- Ensures transceiver PLL placement in core partitions.
- Allows the root_partition to control and merge PLLs, as needed.
- Merging PLLs saves resources and power.
- Simplifies porting the design to new devices or software versions.
- Ensures that PLLs are not hidden in the design.

7.2.5 Block-Based Design Flow

Successful projects that incorporate incremental block-based design start with an iterative design flow:

1. Create your design project.
2. Define the design periphery. This includes specifying which resource types are assigned to the periphery rather than the core. Once you have an initial design framework, click **Processing ➤ Start ➤ Start Analysis & Elaboration**. Analysis & Elaboration populates the Project Navigator **Hierarchy** tab with the project hierarchy.
3. In the **Hierarchy** tab, right-click an entity and click **Design Partition ➤ Set as Design Partition** for each entity that needs partitioning.
   You can modify design partition settings subsequently with the **Design Partitions Window** which is available from the right-click menu or the **Assignments** menu.
4. To compile the design, click **Compile Design** on the Compilation Dashboard.
5. Verify your design results in the Compilation Report.
   Analyze partitions for specific results such as timing closure, area, or space. If a partition meets requirements, proceed to Step 6, otherwise modify the partition and re-compile.
6. Preserve partitions that meet requirements in the **Design Partitions Window**. In the **Design Partitions Window** set the **Preservation Level** to **final**.
7. Verify that partitions meet design requirements.
8. Perform a full compilation and generate device programming files, such as SRAM Object Files (`.sof`), and Programmer Object Files (`.pof`).

Figure 72. Incremental Block-Based Design Flow



Related Links

Design Partition Guidelines on page 225

7.2.5.1 Create and Prepare a Top-Level Design

To create and prepare a top-level design for block-based compilation:

1. Create the top-level project. The top-level project incorporates the entire team-based design and includes the top-level entity that instantiates entities your design requires in partitions developed as separate Quartus Prime projects.
2. Add design files to define the hierarchy of partitions. If the source files are not complete, create a wrapper file to define the port directions in the module or entity.
3. Click **Processing ➤ Start ➤ Start Analysis & Elaboration**.
4. Create assignments that apply to the entire design, including the device, pin location, and timing assignments.
5. In the Project Navigator, right-click any entity to **Set as Design Partition** for each entity that you want to maintain as a separate Quartus Prime project.
6. For each design partition that you maintain as a separate project, or that is not yet complete, in the Design Partitions window, set the **Empty** to **Yes**.
7. Optionally, create a LogicLock Plus region constraint for each partition that you maintain as a separate Quartus Prime project.

   Note: Creating fixed and locked LogicLock Plus regions avoids scattered or overlapping Fitter placement of partitions in the top-level design.
8. On the Compilation Dashboard, click **Start Compilation**.

7.2.5.2 Create or Remove a Partition

The Project Navigator displays a list of entities in the **Hierarchy** tab.

To edit design partitions in the Project Navigator:

1. Click **Processing ➤ Start ➤ Start Analysis & Elaboration**.
2. In the Project Navigator, right-click an instance in the **Hierarchy** tab, click **Design Partition ➤ Set as Design Partition**. A design partition icon appears next to each instance that is set as a partition.
3. To edit or remove an existing design partition, click **Assignments ➤ Design Partitions Window**.

7.2.5.3 Set or Modify Partition Preservation Level

To modify the preservation partition level:

1. Click **Assignments ➤ Design Partitions Window**.
2. Double-click the **Preservation Level** column for a partition.
3. Choose the appropriate preservation level, **source**, **synthesized**, or **final**, depending on the level of information you want to preserve for the next compilation.
   The preservation level can only be set to the level that exists in the current project.

7.3 Design Block Reuse

Design block reuse allows you to preserve a design partition as an exported `.qdb` file, and import this partition into another design project. Reuse of core and periphery design blocks involves partitioning and constraining the block prior to compilation,

export, and reuse. Effective design block reuse requires careful planning to ensure that the source code and design hierarchy support the physical partitioning of device resources that these flows require.

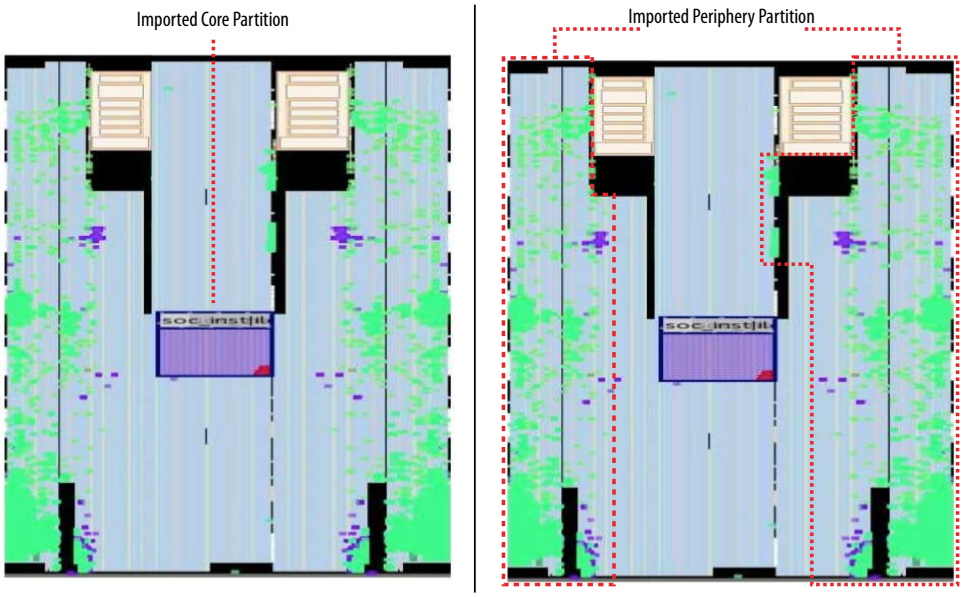
- **Core partition reuse**—allows reuse of design modules with specific optimized functionality or algorithms, such as modules for encryption, encoding, image processing, or other functions in another project.
- **Periphery partition reuse**—allows reuse of a placed and routed periphery (including IO, HSSIO, PCIe, PLLs, as well as core resources), while leaving an empty reconfigurable partition open for subsequent development.

At a high level, the core and device periphery partition reuse flows are similar. Both flows preserve and reuse a design partition as a `.qdb` file. You define, compile, and preserve the block in a "developer project", and then import the block for reuse in one or more "consumer projects."

7.3.1 Design Block Reuse Examples

You can save time by reusing design blocks for the same periphery interface, or for replication of placed and routed IP. You can design, implement, and verify core or periphery blocks just once, and then reuse those blocks multiple times across different projects.

Figure 73. Design Block Reuse Examples



In a typical periphery preservation, you can design an FPGA interface that several projects can use over time. Each project targets the same FPGA part number, and has the same interfaces. Only the dynamic area of the project that contains custom logic changes between projects.

Figure 74. Periphery Reuse on Project with Same Interfaces

In the following example, the periphery is reused across multiple projects. Only the encryption engine and card specific manufacturing data changes between projects.

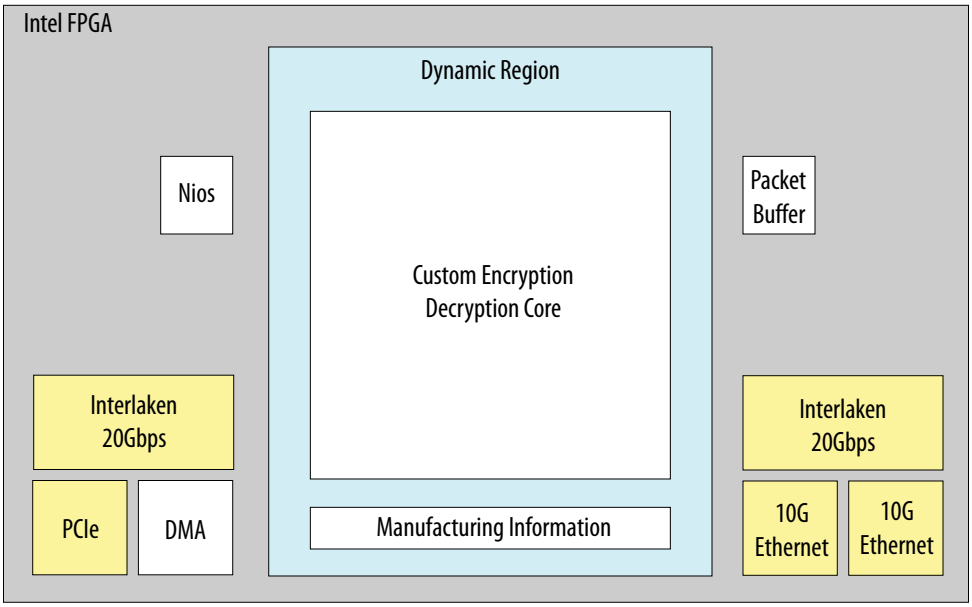
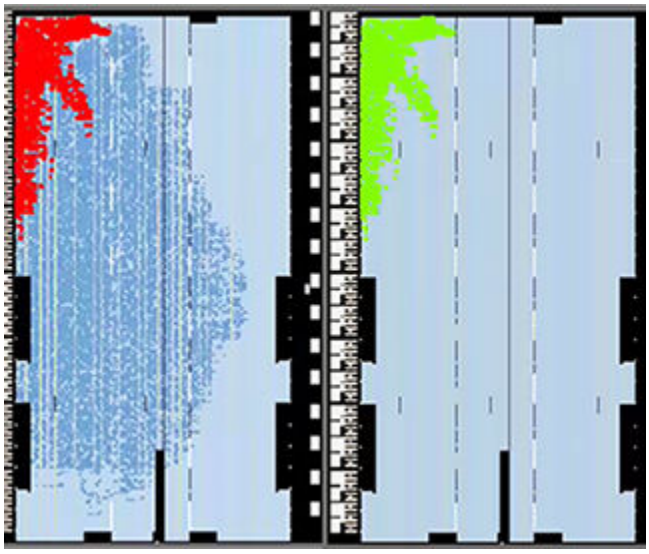


Figure 75. IP Replication and Physical Implementation

You can preserve a block with difficult to close timing or other unique characteristics, and then replicate that functionality and physical implementation in other projects.

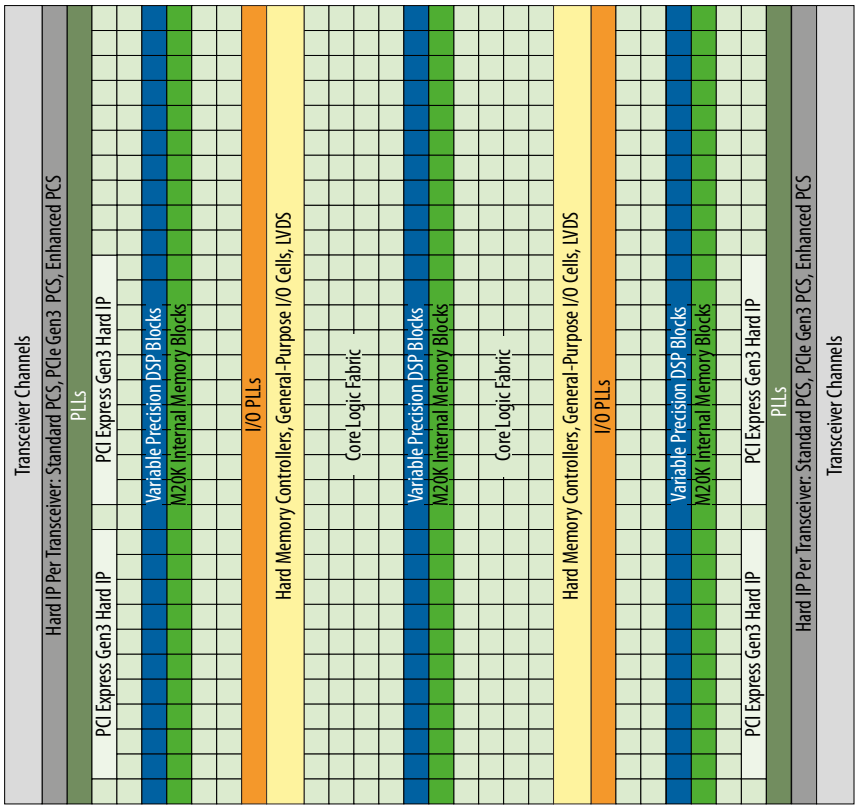


7.3.2 Identifying Blocks for Reuse

When designing for block reuse, you must first determine the logical hierarchy boundaries that you can define as partitions, and the compilation stage to preserve in each case. Set up a design hierarchy and source code to support this partitioning.

Implement core partitions only in device core resources, such as LABs, embedded memory modules (M20Ks and MLABs), and DSP blocks. Implement all periphery partitions, such as transceivers, external memory interfaces, GPIOs, and I/O receivers, in device periphery resources. Use any Quartus Prime-supported design entry method to create your design. For example, you can use Qsys Pro, DSP Builder, or standard design entry languages (SystemVerilog, Verilog HDL, and VHDL) for design entry.

Figure 76. Available Resource Types in Arria 10 Devices



7.3.3 Design Block Reuse Flows

The Quartus Prime software supports the following separate reuse flows for core and periphery design partitions. This chapter describes these flows in detail.

Figure 77. Core Partition Reuse Flow

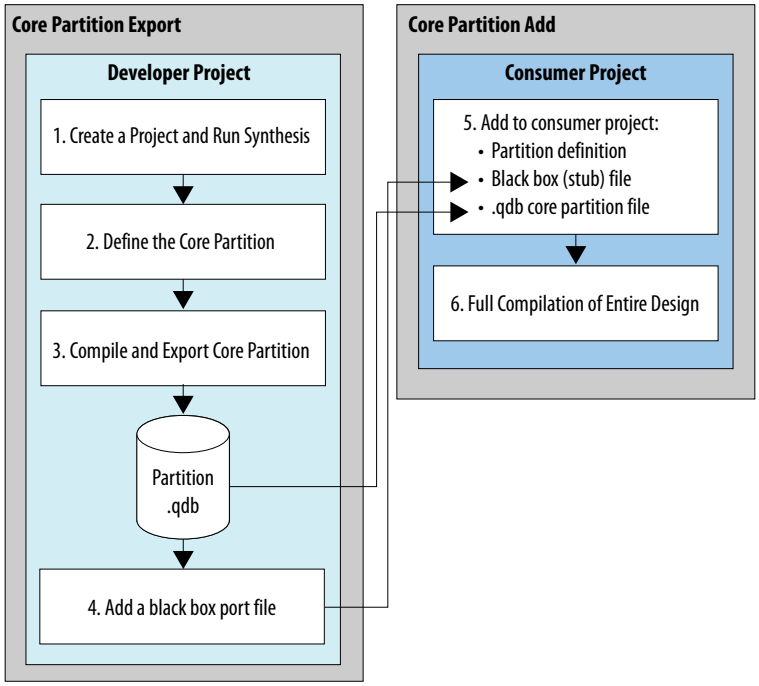


Figure 78. Periphery Partition Reuse Flow

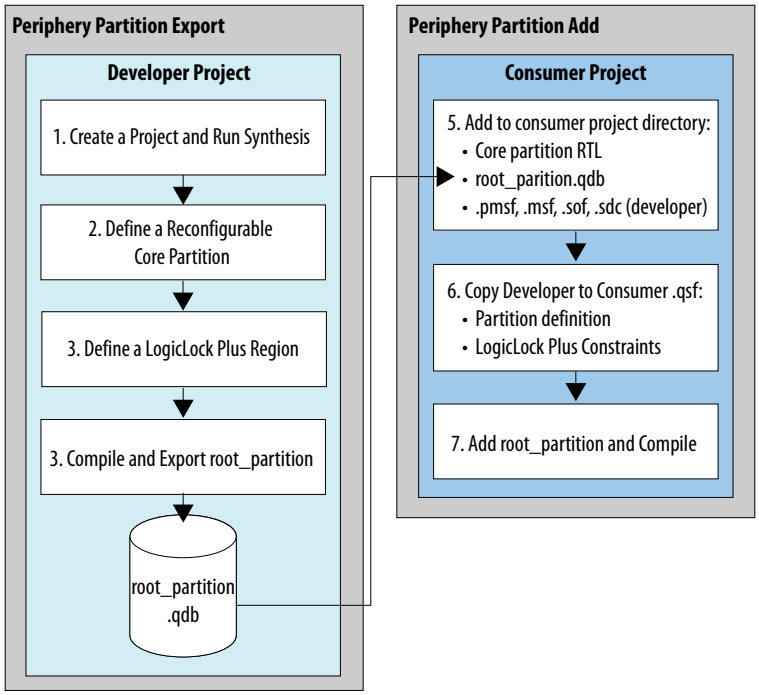


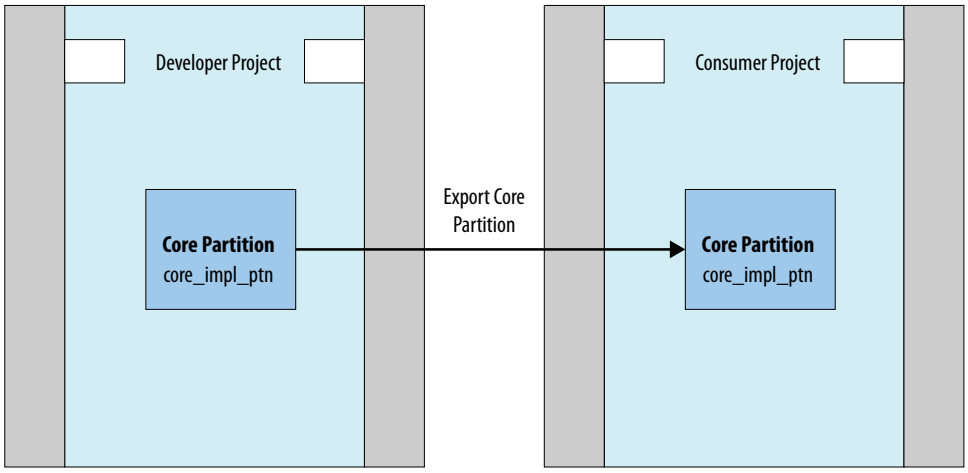
Table 65. Core and Periphery Reuse Comparison

		Core Reuse	Periphery Reuse
Periphery Preservation	Developer	None	Final snapshot exported to `.qdb` file
	Consumer		Periphery reused in project
Core Preservation	Developer	Core partition exported	None
	Consumer	Imported to project	None
Partition	Developer	Partition required	Required for the core partition
	Consumer	Partition required	
Black Box File	Developer	Not required	Not required
	Consumer	Yes	Not required
Programming Files	Developer	Creates `.sof` file	Creates `.msf`, `.pmsf`, `.sof`
	Consumer	Creates `.sof` file	Uses `.msf`, `.pmsf`, `.sof`
Revision Type	Developer	Not required	Partial Reconfig - Base
	Consumer		
.sdc File	Developer	Required for development	Required for development
	Consumer	Not required. Use for analysis	Highly recommended

7.3.4 Reusing Core Partitions

Reusing core partitions involves running design synthesis, exporting a core partition, and importing the core partition into the consumer project. After exporting the core partition to a `.qdb`, only analysis occurs in the consumer project. Changes to an exported partition can only occur in the developer (source) project.

Figure 79. Core Partition Reuse Model



The following sections describe each step in the core partition reuse flow in detail.

7.3.4.1 Step 1: Create a Project and Run Synthesis

Reuse of core partitions requires that you first create and synthesize a representative project that defines the core partition for export.

To setup and synthesize a project for core partition export:

1. Click **File > New Project Wizard** to create a new project and specify a top-level project entity and design file.
2. Create your design source files, and then click **Project ➤ Add/Remove Files In Project** to add the source files.
3. To run design synthesis, click **Analysis & Synthesis** on the Compilation Dashboard. The Compiler synthesizes the design and preserves the Synthesis snapshot.

7.3.4.2 Step 2: Define a Core Partition

Define design partitions to subdivide placement of core design instances along logical boundaries. Confine each core instance for export within a design partition. Partition design instances from the Project Navigator or in the Design Partitions Window.

Follow these guidelines when defining design partitions for reuse:

- Register partition boundary ports. This practice can reduce delay penalties on signals that cross partition boundaries. Also, this technique keeps register-to-register timing paths in a single partition for optimization.
- Minimize the timing-critical paths passing in or out of design partitions. In the case of timing critical-paths that cross partition boundaries, rework the partition boundaries to avoid these paths.
- Avoid generating reset or clock signals inside core partitions. Such signals cannot drive to global networks unless the signal drives out of these core partitions into the root partition, where you instantiate the clock buffer.

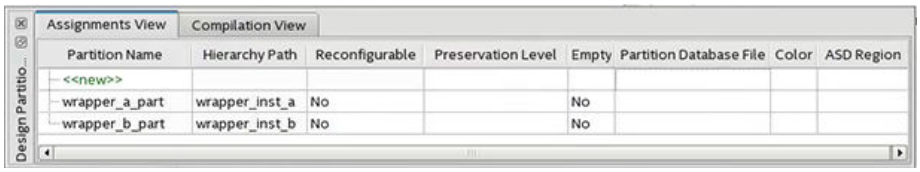
To define a core design partition:

1. Review the project to determine design elements suitable for reuse, and the appropriate snapshot for export.
2. Click **Assignments ➤ Design Partition Window**. Alternatively, open the Design Partitions Window by right-clicking a design instance and selecting **Design Partition ➤ Design Partitions Window**.
3. Double-click the **<<new>>** button in the **Partition Name** column.
4. Select the design instance to partition and click **OK**.
5. Ensure that the **Reconfigurable** option is set to **No**.

   The **Color** column indicates the color of each partition. This color is identical to the partition color in the Chip Planner. Right-click a partition in the window to perform various tasks, such as deleting the partition, locating the node, or creating LogicLock Plus regions for the partition.

   The Quartus Prime software automatically generates a partition name, based on the entity name and hierarchy path. If the project uses the entity more than once, each partition name is incremented with a number. Edit the partition name in the Design Partitions Window by double-clicking the name.

Figure 80. Design Partitions Window



7.3.4.3 Step 3: Compile and Export the Core Partition(s)

This step describes running the Compiler to generate a snapshot of a core partition for export. The Compiler preserves the results of each stage as a snapshot for analysis or reuse. Following compilation, you can export a fully placed and routed snapshot of the core partition.You can reuse the exported partition in the same project or in another project. When you compile the exported partition in a consumer project, the partition only Compiles through stages not already run in the developer project.

1. Click **Processing ➤ Start ➤ Start Fitter** to run all compilation stages through fitting. Alternatively, run any of the following Compiler stages on the Compilation Dashboard:
   - To perform periphery placement and routing, click **Plan**.
   - To begins assigning core logic to device resources, click **Early Place**.
   - To complete core logic placement, click **Place**.
   - To fully route the design, click **Route**.
   - To complete all compilation processing, click **Fitter (Finalize)**.
2. To export the core partition, click **Project ➤ Export Design Partition**. Select the desired **Design Partition** for export, and the **Snapshot** that matches the desired compilation stage.

7.3.4.4 Step 4: Add a Black Box File

Follow these steps to create a block box port definitions file for the partition.

1. Create an HDL file (`.v`, `.vhd`, `.sv`) that contains only the port definitions for the exported core partition. Include parameters passed to the module. For example:

```
module bus_shift #(
    parameter DEPTH=256,
    parameter WIDTH=8
  )(
    input clk,
    input enable,
    input reset,
    input [WIDTH-1:0] sr_in,
    output [WIDTH-1:0] sr_out
  );
endmodule
```

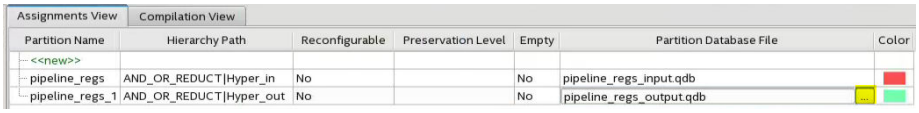
2. Copy the black box file and core partition `.qdb` file to the consumer project directory. To check timing results of the core partition in the consumer project, copy any `.sdc` file for the module to the project directory. You can use the `.sdc` file to verify the timing of the fully implemented design.

7.3.4.5 Step 5: Add the Core Partition and Compile

To add the core partition to a consumer project, you add the black box as a source file, and assign the core partition `.qdb` to an instance in the Design Partitions Window. Because the exported `.qdb` includes place and route information, the consumer project device must be identical to the design project device. The consumer project must supply a clock and any other constraints required for the interface to the exported core partition.

1. Create or open a Quartus Prime project to consume the core partition.
2. To add the black box and `.qdb` files to the consumer project, click **Project ➤ Add/Remove Files in Project** and select these files.
3. To set the top-level entity, right-click the file in the Project Navigator and select **Set as Top-Level Entity**.
4. To run design synthesis, click **Analysis & Synthesis** on the Compilation Dashboard.
5. To create a design partition for the black box file, right-click the black box instance in the Project Navigator, and then click **Design Partition ➤ Set As Design Partition**.
6. To assign a `.qdb` file to an instance in the Design Partitions Window, click **Assignments ➤ Design Partitions Window**. In the **Partition Database File** column, select the `.qdb` file for that entity.

Figure 81. Design Partitions Window



7. Click **Processing ➤ Start ➤ Start Fitter** to run all compilation stages through fitting.
8. The Fitter generates Partition Summary reports that list details about any partitions in your project, such as the partition name, hierarchy path, snapshot preservation level, and any associated `.qdb` file.

Figure 82. Partition Summary Report

Fitter Partition Summary

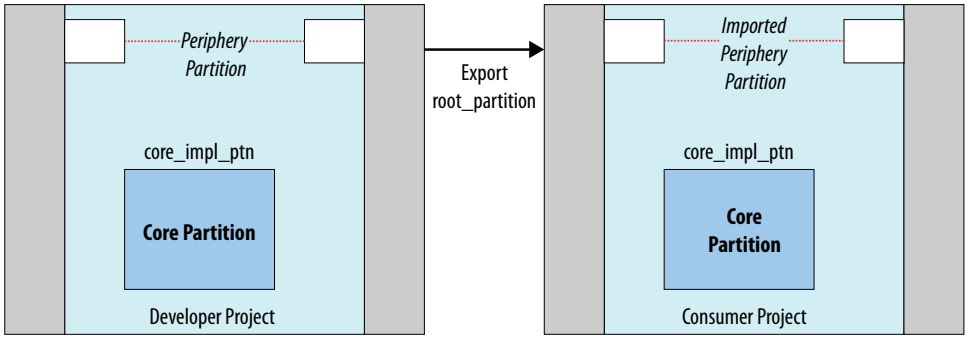
<<Filter>>

	Partition Name	Hierarchy Path	Preservation	Empty	QDB File
1	root_partition	|			
2	pipeline_regs	AND_OR_R...Hyper_in	final		pipeline_regs_input.qdb
3	pipeline_regs_1	AND_OR_R...yper_out	final		pipeline_regs_output.qdb

7.3.5 Reusing Periphery Partitions

Reuse of device periphery blocks allows you to design an FPGA to board interface and associated logic once, and then replicate that periphery in other projects. Periphery partition reuse requires two distinct projects. You develop the periphery in one project, and export the root partition. Another project can then consume the exported root partition.

Figure 83. Periphery Partition Reuse Model



The periphery developer first plans all periphery resources and associated logic in the developer project. The periphery developer leaves one or more partitions for subsequent implementation of core logic in the consumer project. The following sections describe each step in the core partition reuse flow in detail.

7.3.5.1 Step 1: Create a Project and Run Synthesis

Reuse of periphery partitions requires that you first create and synthesize a representative project that defines the core and periphery partitions for export, and reserve a reconfigurable partition to contain core logic in the consumer project. The project revision type must be **Partial Reconfiguration - Base**, to indicate to the Compiler that the revision contains an empty partition.

To setup and synthesize a developer project for periphery partition export:

1. Click **File > New Project Wizard** to create a new project and specify a top-level project entity and design file.
2. To create a project revision, click **Project ➤ Revisions**. For **Revision Type**, select **Partial Reconfiguration - Base**.
3. To run design synthesis, click **Analysis & Synthesis** on the Compilation Dashboard. The Compiler synthesizes the design.

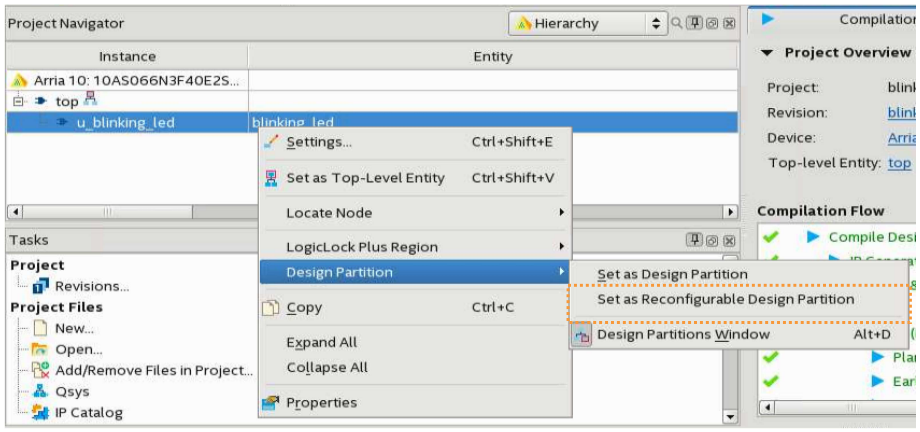
7.3.5.2 Step 2: Create a Reconfigurable Design Partition

To export and reuse periphery modules, first create a reconfigurable design partition to contain the core partition. This partition reserves device resources for core logic when you import the periphery to the consumer project.

To assign a reconfigurable design partition:

1. Click the **Hierarchy** tab in the Project Navigator.
2. Right-click the core partition in the **Instance** list and click **Design Partition ➤ Set as Reconfigurable Design Partition**.

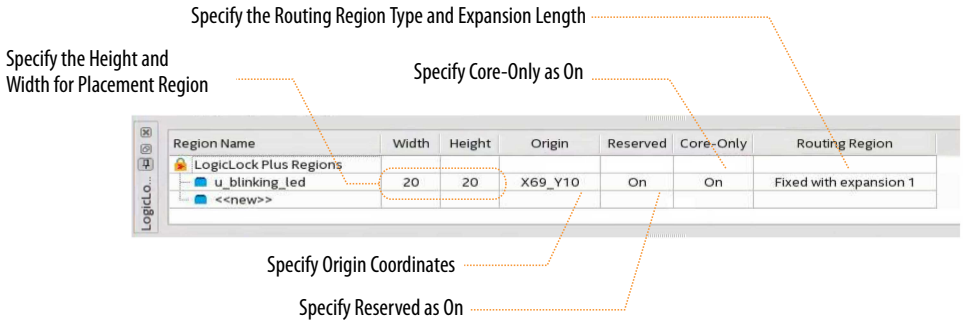
Figure 84. Creating Reconfigurable Design Partitions from Project Navigator



7.3.5.3 Step 3: Define a LogicLock Plus Region

Define a core-only, reserved, fixed routing region to reserve core resources in the consumer project for non-periphery development. Assign all core logic inside the core-only LogicLock Plus region. Ensure that the exclusive placement region size can contain all core logic. For projects with multiple core partitions, constrain each partition in a non-overlapping routing region.

Figure 85. LogicLock Plus Regions Window



Follow these steps to define a core-only, reserved, fixed routing region to reserve core resources in the consumer project for non-periphery development:

1. Right-click the design instance in the **Project Navigator** and click **LogicLock Plus Region ➤ Create New LogicLock Plus Region**. The region appears on the LogicLock Plus Regions Window.
2. Specify the placement region co-ordinates in the **Origin** column.
3. Enable the **Reserved** and **Core-Only** options.
4. Click the **Routing Region** cell. The **LogicLock Plus Routing Region Settings** dialog box appears.
5. Specify **Fixed Width Expansion** with **Expansion Length** of **0** for the **Routing Type**.
6. Click **OK**.
7. Click **File ➤ Save Project**.

7.3.5.4 Step 4: Compile and Export the Periphery Partition

This step describes running stages of the Compiler to generate a snapshot of the periphery partition for export.

Follow these steps to generate a routed snapshot of the periphery partition for export:

1. On the Compilation Dashboard, click **Fitter (Finalize)** to run all compilation stages through the Final stage.
2. The Quartus Prime GUI does not yet support export of the periphery `root_partition.qdb`. To export the `root_partition.qdb`, type the following command:

```
quartus_cdb <project name> -c <revision name> --export_partition
"root_partition"
    --snapshot final --file root_partition.qdb --exclude_pr_subblocks
```

   This command exports the fully placed and routed periphery partition as the root partition. The `--exclude_pr_modules` option excludes reconfigurable partitions from being exported with the periphery. The area and resource in the core LogicLock Plus region is reserved for core implementation logic.

7.3.5.5 Step 5: Add Files and Constraints

After exporting the periphery partition (`root_partition.qdb`), copy this and other generated files to the consumer project directory. To ensure consistency and save time, copy the partition definition and LogicLock Plus constraints from the developer project `.qsf` file into the consumer project `.qsf` file.

1. Copy the following files to the consumer project directory:
   - `root_partition.qdb`—contains final compilation results for the exported periphery partition. The file name must be `root_partition.qdb`.
   - <*project*>.<*core partition*>`/output_files/.pmsf`—mask file to verify the bit settings.
   - <*project*>`/output_files/`<*file*>`.static.msf`—verifies the bit masks in the assembled .pof file.
   - <*project*>`/output_files/`<*file*>`.sof`—periphery partition programming image integrates with consumer project programming image file.
   - <*file*>`.sdc`—periphery partition timing constraints.

   Note: Although programming file generation does not require the `.pmsf`, `.msf`, `.sof` files, the Assembler reports an error if these files are missing from the consumer project.
2. Copy the LogicLock Plus constraints and partition definition from the developer project `.qsf` into the consumer project `.qsf`. The following shows an example of LogicLock Plus constraints and partition definition:

```
set_instance_assignment -name PARTITION blinking_led -to u_blinking_led -
entity top
set_instance_assignment -name PARTIAL_RECONFIGURATION_PARTITION ON -to
u_blinking_led -entity top

set_instance_assignment -name PLACE_REGION "X63 Y102 X185 Y162" -to
u_blinking_led
```

```
set_instance_assignment -name ROUTE_REGION "X63 Y102 X185 Y162" -to
u_blinking_led
set_instance_assignment -name RESERVE_PLACE_REGION ON -to u_blinking_led
set_instance_assignment -name CORE_ONLY_PLACE_REGION ON -to u_blinking_led
set_instance_assignment -name REGION_MEMBER ON -from u_blinking_led -to
u_blinking_led
```

7.3.5.6 Step 6: Add the Periphery Partition and Compile

Add the `root_partition.qdb` and the core partition RTL as source files in the consumer project. Placement and routing of the imported periphery partition is identical in the consumer project periphery area. This placement constraint excludes clocks, because they may be global signals.

Follow these steps to import the periphery partition to the consumer project:

1. Create or open a Quartus Prime project to consume the exported periphery partition.
2. To add the `root_partition.qdb` to the project, click **Project ➤ Add/Remove Files in Project**. Select the `root_partition.qdb` file, click **Add**, and then click **OK**.
3. To add any source file for the reconfigurable core partition, click **Project ➤ Add/Remove Files in Project** and add the file(s).
4. To set the project revision type, click **Project ➤ Revisions**. In **Revision Type**, select **Partial Reconfiguration - Base**.
5. On the Compilation Dashboard, click **Compile Design** to run all compilation stages.

The Quartus Prime Pro Edition Compiler implements the imported periphery.

7.4 Document Revision History

This document has the following revision history.

Table 66. Document Revision History

Date	Version	Changes
2017.05.08	17.0.0	• First public release.

8 Creating a Partial Reconfiguration Design

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

You can also include multiple parent and child partitions, or create multiple levels of partitions in your design. This flow, referred to as the hierarchical partial reconfiguration (HPR), includes a static region that instantiates the parent PR region, and the parent PR region instantiating the corresponding child PR region. You can perform the same PR region reprogramming for either the child or the parent partition. Reprogramming a child PR region does not affect the parent or the static region. Reprogramming the parent region reprograms the associated child region with the default child persona, without affecting the static region.

Note: The HPR flow does not impose any restrictions on the number of sub-partitions you can create in your design.

PR provides the following advancements to a flat design:

- Allows run-time design reconfiguration
- Increases scalability of the design through time-multiplexing
- Lowers cost and power consumption through efficient use of board space
- Supports dynamic time-multiplexing functions in the design
- Improves initial programming time through smaller bitstreams
- Reduces system down-time through line upgrades
- Enables easy system update by allowing remote hardware change

The Quartus Prime Pro Edition software supports the PR feature for the Arria 10 device family.

Table 67. Partial Reconfiguration Feature Advancements for Quartus Prime Pro Edition Software

PR in Quartus Prime Standard Edition Software	PR in Quartus Prime Pro Edition Software
Requires freezing all the non-global inputs of a PR region, except the global clocks.	No requirement to freeze all the non-global PR region inputs.
Limits use to 6 global clocks in each PR region.	Allows using up to 33 global clocks in each PR region.
The maximum PR clock frequency is 62.5 MHz.	The maximum PR clock frequency is 100 MHz, decreasing the programming time.
Requires manually running the Analysis & Synthesis, Fitter, and Assembler Compiler modules.	Automates the process by providing a compilation flow script.

Related Links

- Design Planning for Partial Reconfiguration
  For information on partial reconfiguration for Stratix V devices.
- Partial Reconfiguration IP Core User Guide
  For information on Partial Reconfiguration IP Core and how to instantiate it.
- Arria 10 Device Overview
  For complete information on the Arria 10 device family.
- Arria 10 Reconfiguration Interface and Dynamic Reconfiguration
  For complete information on using the Arria 10 reconfiguration interface that is part of the Transceiver Native PHY IP core and the Transceiver PLL IP cores.
- Partial Reconfiguration Tutorials
  For scripts, reference designs, and tutorials on partial reconfiguration design flow.

8.1 Partial Reconfiguration Concepts

Implementing a PR design requires understanding of the FPGA device capabilities and the Quartus Prime compilation flow. The following table defines common PR terminology:

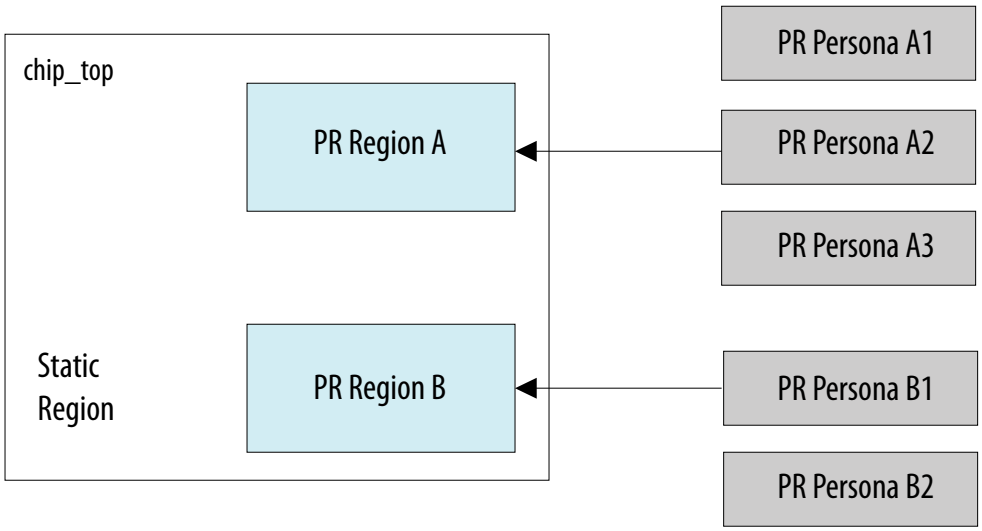
Table 68. Partial Reconfiguration Terminology

Term	Description
PR partition	Design partition you designate for PR. A PR project can contain one or more partially reconfigurable PR partitions.
PR region	An area in the FPGA that you associate with a partially reconfigurable partition. A PR region contains the core locations of the device you wish to reconfigure. A device can contain more than one PR region. A PR region can be core-only, such as LAB, RAM, or DSP.
Static region	All areas outside the PR regions in your project. You associate the static region with the top-level partition of the design. The static region contains both the core and periphery locations of the device.
PR persona	A specific PR partition implementation in a PR region. A PR region can contain multiple personas. Static regions contain only one persona.
PR control block	A dedicated FPGA block. The PR control block processes the PR requests, handshake protocols, and verifies the cyclic redundancy check (CRC).

continued...

Term	Description
PR host	The system for coordinating PR. The PR host communicates with the PR control block. Implement the PR host within the FPGA (internal PR host) or in a chip or microprocessor (external PR host).
PR IP Core	The Altera Partial Reconfiguration IP Core that you instantiate in the static region of your design. This IP core interfaces with the PR control block to manage the bitstream source.
Floorplan	The layout of physical resources on the device. Creating a design floorplan, or floorplanning, is the process of mapping logical design hierarchy to physical regions in the device.
Revision	A collection of settings and constraints for one version of your project. A Quartus Prime Settings File (`.qsf`) preserves each revision of your project. Your Quartus Prime project can contain several revisions. Revision allows you to organize several versions of your design within a single project.
Snapshot	The output of a Compiler stage. The Quartus Prime Pro Edition Compiler generates a snapshot of the compiled database after each stage. Export the snapshot at various stages of the compilation flow, such as synthesis or final.

Figure 86. A Partial Reconfiguration Design



8.2 Partial Reconfiguration Design Flow

The PR design flow requires initial planning. This planning involves setting up the design partition(s), and determining the placement assignments in the floorplan. Well-planned PR partitions improve design area utilization and performance. Your design can include the PR control block, which involves implementing the internal or external PR host.

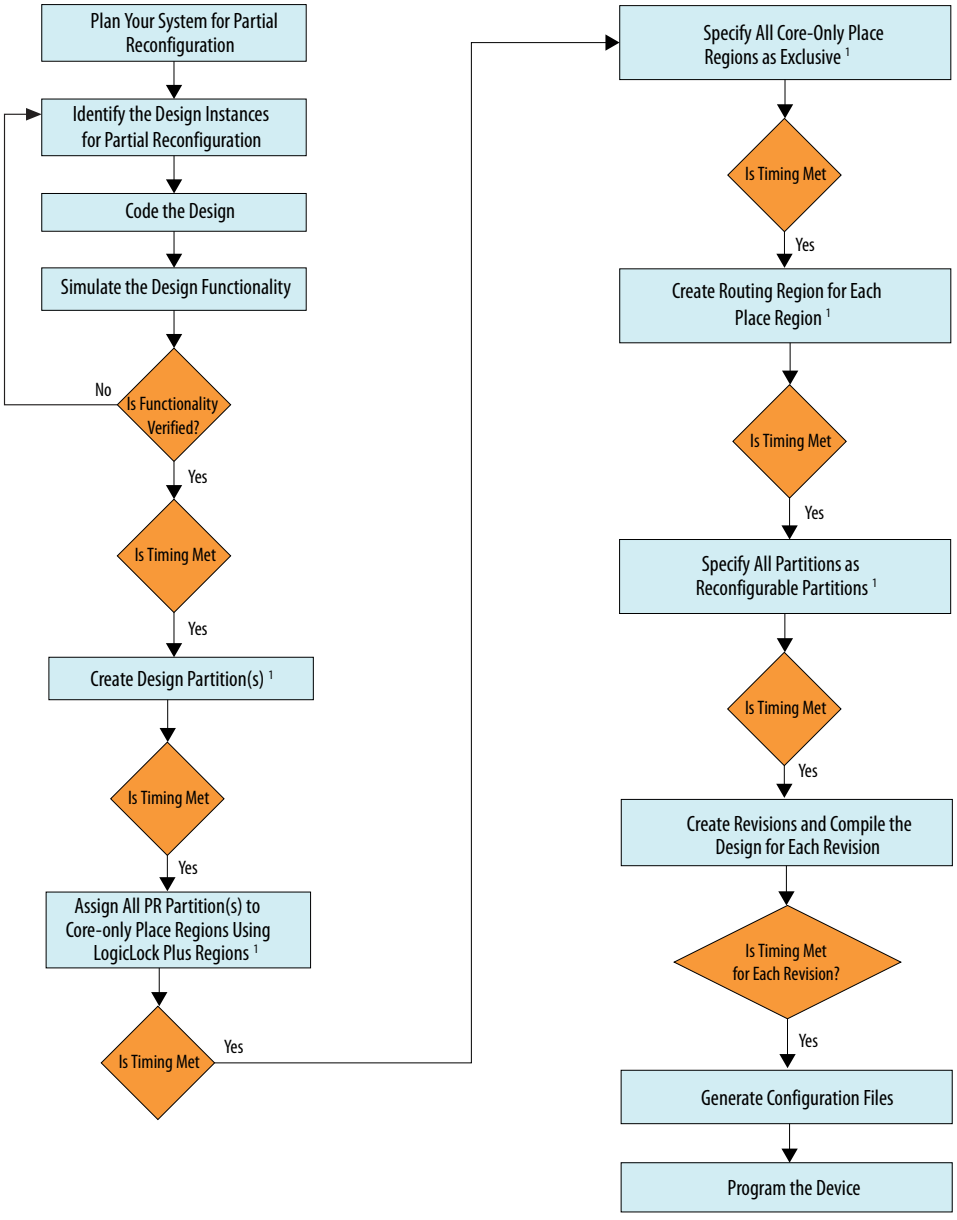
The PR design flow uses the project revisions feature in the Quartus Prime software. Your initial design is the base revision, where you define the static region boundaries and reconfigurable regions on the FPGA. From the base revision, you create multiple revisions. These revisions contain the different implementations for the PR regions. However, all PR implementation revisions use the same top-level placement and routing results from the base revision.

In order to debug your design at a later stage using the Signal Tap Logic Analyzer, you must instantiate the SLD JTAG bridge IP components for each PR region in your design. The SLD JTAG Bridge consists of two IP components - SLD JTAG Bridge Agent and SLD JTAG Bridge Host. Perform the following steps during the early planning stage, to ensure you can signal tap your static as well as PR region, at a later stage:

1. Instantiate the SLD JTAG Bridge Agent IP in the static region.
2. Instantiate the SLD JTAG Bridge Host IP in the PR region of the default persona.
3. Then, instantiate the SLD JTAG Bridge Host IP for each of the personas during the synthesis revision creation for the personas.

For more information on the SLD JTAG bridge instantiation, refer to *Instantiating a SLD JTAG Bridge Agent* and *Instantiating a SLD JTAG Bridge Host* sections in Volume 3 of Quartus Prime Pro Edition handbook.

Figure 87. Partial Reconfiguration Design Flow



(1) Recommended to compile the base revision before verifying timing closure

Related Links

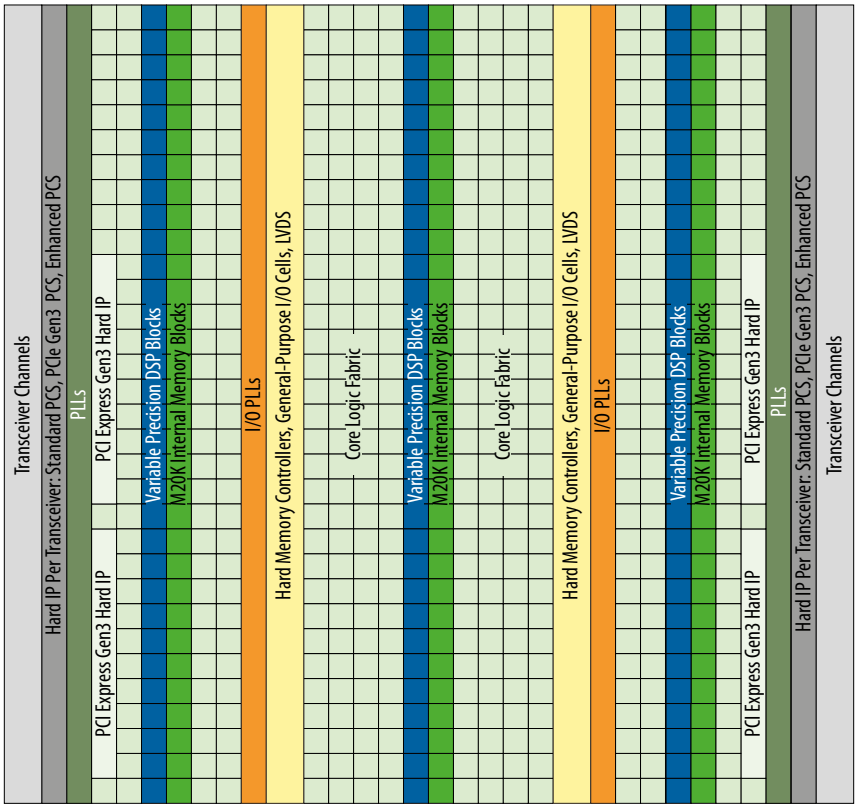
- Instantiating a SLD JTAG Bridge Agent
- Instantiating a SLD JTAG Bridge Host

8.2.1 Identify Resources for Partial Reconfiguration

When designing for partial reconfiguration, first determine the logical hierarchy boundaries that you can define as reconfigurable partitions. Next, set up the design hierarchy and source code to support this partitioning.

Reconfigurable partitions can contain only core resources, such as LABs, embedded memory blocks (M20Ks and MLABs), and DSP blocks in the FPGA. All periphery resources, such as transceivers, external memory interfaces, GPIOs, I/O receivers, and hard processor system (HPS), must be in the static portion of the design. Partially reconfiguration of global network buffers for clocks and resets is not possible.

Figure 88. Available Resource Types in Arria 10 Devices



The following table shows the supported reconfiguration type for each FPGA block in the Arria 10 device:

Table 69. Supported Reconfiguration Methods in Arria 10 Device

Hardware Resource Block	Reconfiguration Method
Logic Block	Partial reconfiguration
Digital Signal Processing	Partial reconfiguration
Memory Block	Partial reconfiguration
Core Routing	Partial reconfiguration

continued...

Hardware Resource Block	Reconfiguration Method
Transceivers	Dynamic reconfiguration
PLL	Dynamic reconfiguration
I/O Blocks	Not supported
Clock Control Blocks	Not supported

Use any Quartus Prime-supported design entry method to create core-only logic for a PR partition. Use Qsys Pro or DSP builder for Intel FPGAs Advanced, in addition to the standard design entry languages, such as SystemVerilog, Verilog HDL, and VHDL, for design entry.

The following IP cores support system-level debugging in the static region:

- In-System Memory Content Editor
- In-System Sources and Probes Editor
- Virtual JTAG
- Nios® II JTAG Debug Module
- Signal Tap Logic Analyzer

Note: Only Signal Tap Logic Analyzer allows simultaneous debugging of the static and PR regions.

Related Links

- Arria 10 Device Overview
  For complete information on the Arria 10 device family.
- Arria 10 Reconfiguration Interface and Dynamic Reconfiguration
  For complete information on using the Arria 10 reconfiguration interface that is part of the Transceiver Native PHY IP core and the Transceiver PLL IP cores.

8.2.2 Create Design Partitions for Partial Reconfiguration

Create design partitions for each PR region that you want to partially reconfigure. You can create any number of independent partitions or PR regions in your design. Create design partitions for partial reconfiguration from the Project Navigator, or using the Design Partitions Window.

A design partition is the logical partitioning of the design, and does not specify a physical area on the device. Associate the partition with a specific area of the FPGA using LogicLock Plus Region floorplan assignments. To avoid partitions obstructing design optimization, group the logic together within the same partition. If your design includes a hierarchical PR flow with parent and child partitions, you can assign multiple parent or child partitions to your design, as well as multiple levels of PR partitions.

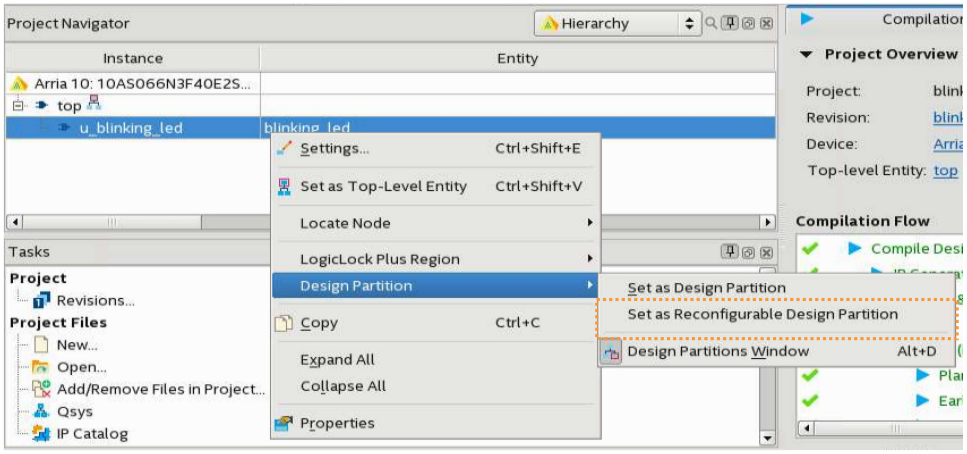
Good hierarchical design practices result in a successful partial reconfiguration FPGA design. Follow these guidelines when creating partitions for PR regions in your design:

- Register all the partition boundaries, and all the inputs and outputs of each partition.
- Minimize the number of paths crossing the partition boundaries.
- Minimize the timing-critical paths passing in or out of the PR regions. In case of timing critical-paths crossing the PR region boundaries, rework the PR regions to avoid these paths.
- Avoid creating reset or clock signals inside the PR regions.

To create design partition for the partial reconfiguration project from the Project Navigator:

1. Click the **Hierarchy** tab in the Project Navigator.
2. Right-click the entity in the **Instance** list and click **Design Partition ➤ Set as Reconfigurable Design Partition**.

Figure 89. Creating Reconfigurable Design Partitions from Project Navigator

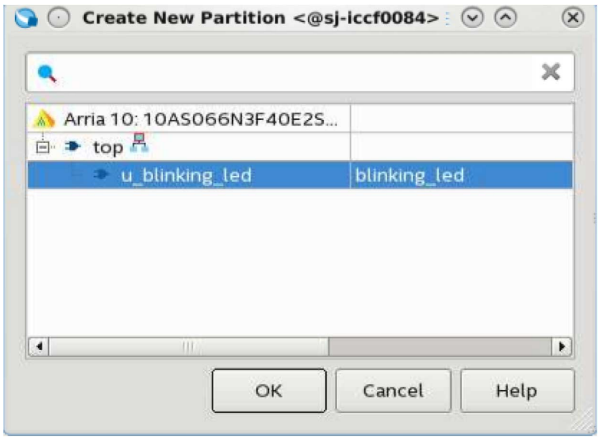


To create a design partition for the partial reconfiguration project from the Design Partition Window:

1. Click **Assignments ➤ Design Partition Window**.

   Note: Alternatively, open the Design Partitions Window by right-clicking the design instance and selecting **Design Partition ➤ Design Partitions Window**.
2. Double-click the **<<new>>** button in the **Partition Name** column.
3. Select the design instance to partition and click **OK**.

Figure 90. Create New Partition Window



4. Double-click the **Reconfigurable** option to select **Yes**.

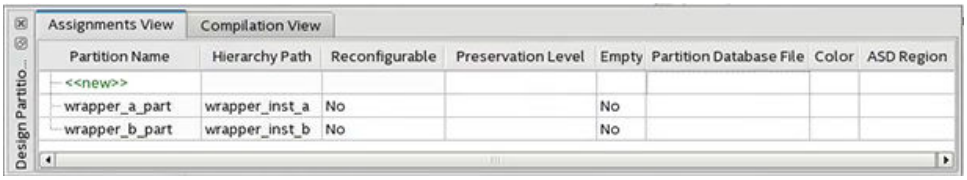
The **Color** column indicates the color of each partition. This color is identical to the partition color in the Quartus Prime Design Partition Planner and Chip Planner. Right-click a partition in the window to perform various tasks, such as deleting the partition, locating the node, or creating LogicLock Plus regions for the partition.

The Quartus Prime software automatically generates a partition name, based on the instance name and hierarchy path. This default partition name varies with each instance. Edit the partition name in the Design Partitions Window by double-clicking the name.

The following assignments in the `.qsf` file correspond to the design partition creation in the Design Partitions Window:

```
set_instance_assignment -name PARTITION pr_partition -to <design_instance>
set_instance_assignment -name PARTIAL_RECONFIGURATION_PARTITION ON -to
<design_instance>
```

Figure 91. Design Partitions Window



Note: The current version of the Quartus Prime Pro Edition software does not support the creation of nested PR partitions.

Related Links

Creating Design Partitions

8.2.3 Define Personas

Your partial reconfiguration design can have multiple PR partitions, each with multiple personas. Each of these personas function differently. However, all the PR personas must use the same set of signals to interact with the static region. Ensure that the signals interacting with the static region are a super-set of all the signals in all the personas. A PR design requires identical I/O interface for each persona in the PR region.

8.2.3.1 Create Wrapper Logic for PR Regions

Create the wrapper logic to ensure that all the personas appear similar to the static region. Define a wrapper for each persona, and instantiate the persona logic within the wrapper. In this wrapper, you can create dummy ports to ensure that all the personas of a PR region have the same connection to the static region.

During the PR compilation, the Compiler converts each of the non-global ports on interfaces of the PR region into boundary port wire LUTS. The naming convention for boundary port wire LUTs are `<input_port>~IPORT` for input ports, and `<output_port>~OPORT` for output ports. For example, the instance name of the wire LUT for an input port called `my_input`, on a PR region named `my_region`, is `my_region|my_input~IPORT`.

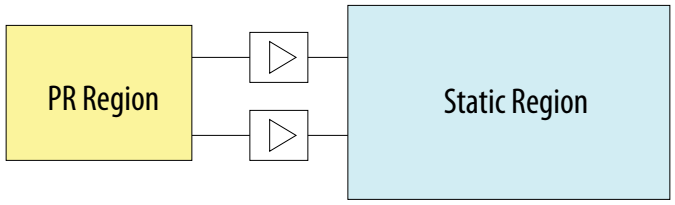
Manually floorplan the boundary ports using the LogicLock Plus region assignments, or place the boundary ports automatically using the Fitter. The Fitter places the boundary ports during the base revision compile. The boundary LUTs are invariant locations, based on the persona you compile. These LUTs represent the boundaries between the static and the PR routing and logic. The placement remains stationary regardless of the underlying persona, because the routing from the static logic does not vary with a different persona implementation.

To constrain the all boundary ports within a given region, use a wildcard assignment. For example:

```
set_instance_assignment -name PLACE_REGION "65 59 65 85" -to u_my_top|design_inst|
pr_inst|pr_inputs.data_in*~IPORT
```

This assignment constrains all the wire LUTS corresponding to the IPORTS specified within the place region, between the coordinates (65 59) and (65 85).

Figure 92. Wire-LUTs at the PR Region Boundary



Optionally, floorplan the boundary ports down to the LAB level, or individual LUT level. To floorplan to the LAB level, create a 1x1 LogicLock Plus `PLACE_REGION` constraint (single LAB tall and a single LAB wide). Optionally, specify a range constraint by creating the desired LogicLock Plus placement region that spans the desired range. For more information on floorplan assignments, refer to *Floorplan the Partial Reconfiguration Design*.

Related Links

Floorplan the Partial Reconfiguration Design on page 268
For more information on floorplanning your design.

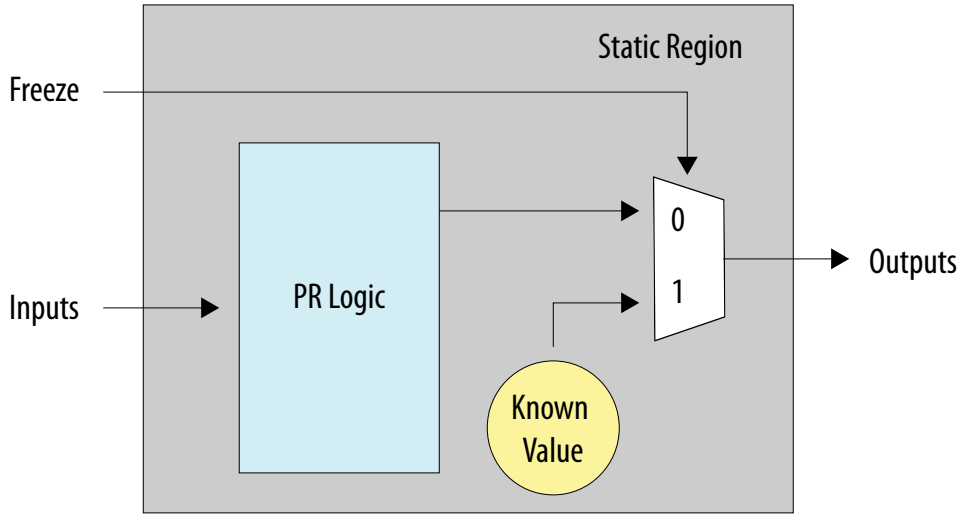
8.2.3.1.1 Freeze Logic for PR Regions

When partially reconfiguring a design, freeze all the outputs of each PR region to a known constant value. This freezing prevents the signal receivers in the static region from receiving undefined signals during the partial reconfiguration process.

The PR region cannot drive valid data until the partial reconfiguration process is complete, and the PR region is reset. Freezing is mainly important for control signals that you drive from the PR region.

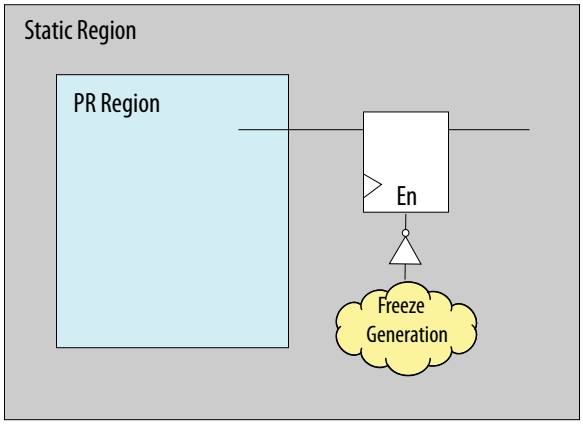
The freeze technique is specific to a PR design. The freeze logic must reside in the static region of your design. A common freeze technique is to instantiate 2-to-1 multiplexors on each output of the PR region, to hold the output constant during partial reconfiguration.

Figure 93. Freeze Technique #1 for Arria 10 Devices



An alternative freeze technique is to register all outputs of the PR region in the static region. Then, use an enable signal to hold the output of these registers constant during partial reconfiguration.

Figure 94. Freeze Technique #2 for Arria 10 Devices



Note: For Arria 10 devices, there is no requirement to freeze the global and non-global inputs of a PR region.

The Partial Reconfiguration Region Controller IP core includes a freeze port for the region that it controls. Include this IP component with your system-level control logic to freeze the PR region output. For designs with multiple PR regions, instantiate one PR Region Controller IP core for each PR region in the design.

The static region logic must be independent of all the outputs from the PR regions for a continuous operation. Control the outputs of the PR regions by creating an RTL wrapper around the PR region.

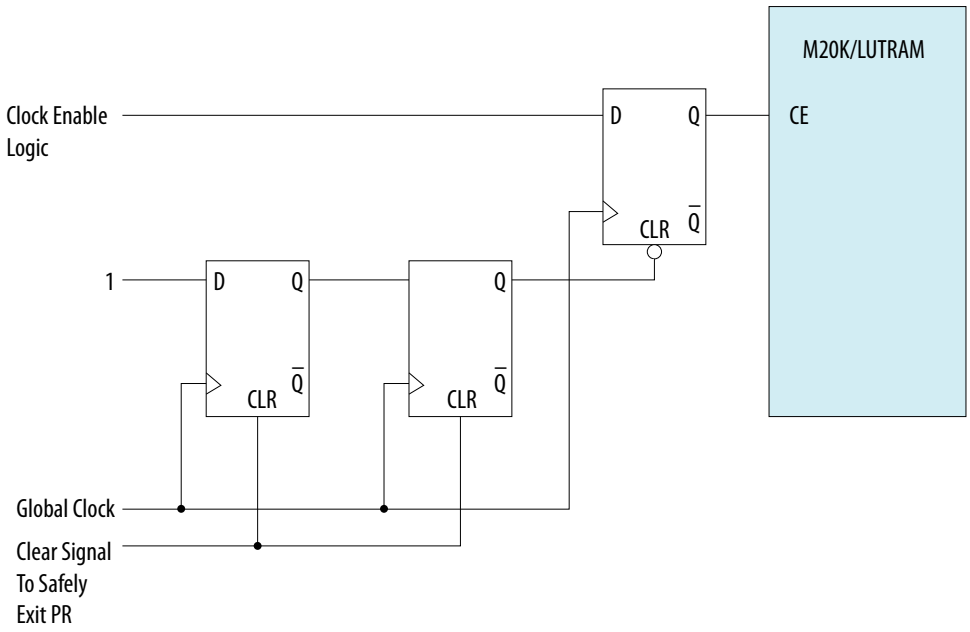
Related Links

Partial Reconfiguration IP Solutions User Guide

8.2.3.2 Implement Clock Enable for On-Chip Memories with Initialized Contents

To avoid spurious writes during PR programming for memories with initialized contents, implement the clock enable circuit in the same PR region as the M20K or MLAB RAM. This circuit depends on an active-high clear signal from the static region. Before you begin the PR programming, assert this signal to disable the memory's clock enable. Your system PR controller must deassert the clear signal on PR programming completion.

Figure 95. RAM Clock Enable Circuit for PR Region



Example 70. Verilog RTL for Clock Enable

```
reg ce_reg;
reg [1:0] ce_delay;

always @(posedge clock, posedge freeze) begin
    if (freeze) begin
        ce_delay <= 2'b0;
    end
    else begin
        ce_delay <= {ce_delay[0], 1'b1};
    end
end

always @(posedge clock, negedge ce_delay[1]) begin
    if (~ce_delay[1]) begin
        ce_reg <= 1'b0;
    end
    else begin
        ce_reg <= clken_in;
    end
end

wire ram_wrclocken;
assign ram_wrclocken = ce_reg;
```

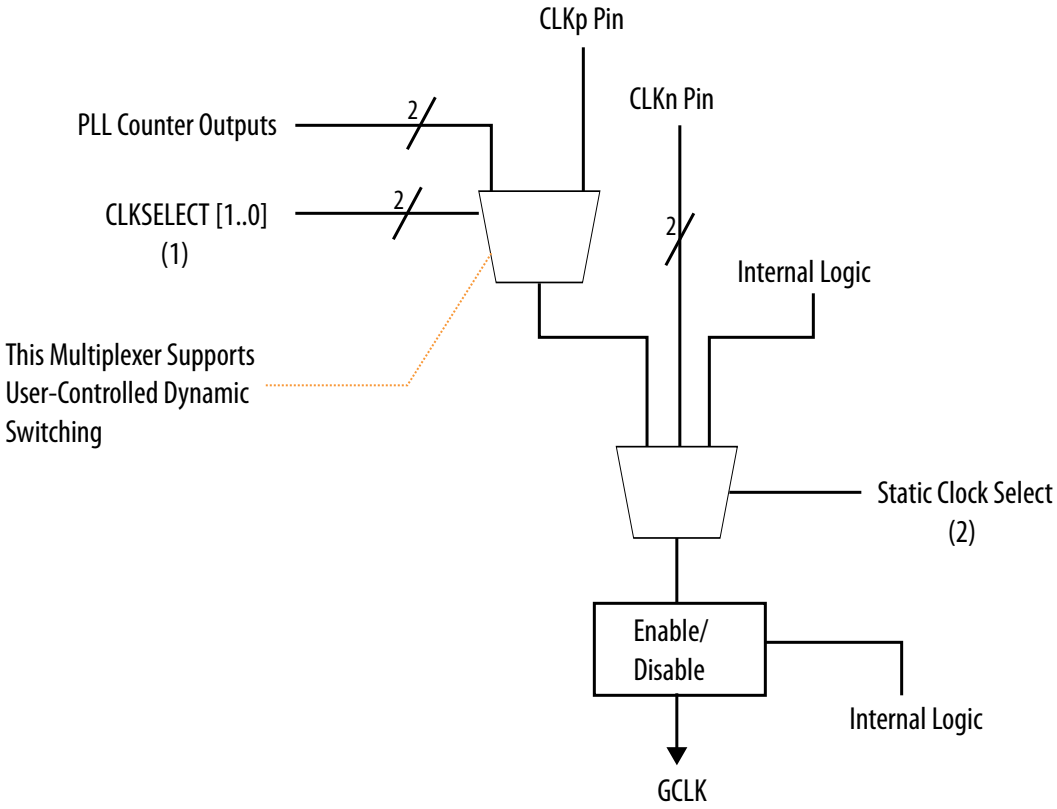
8.2.3.2.1 Clock Gating

An alternate method to avoid spurious writes of initialized content memories is to gate the clock feeding the memories in the static region of your design.

Clock gating is logically equivalent to using clock enable on the memories. This method provides the following features:

- Uses the enable port of the global clock buffers to disable the clock before starting the partial reconfiguration operation.
- Enables the clock on PR completion.
- Ensures that the clock does not toggle during reconfiguration, and requires no additional logic to avoid spurious writes.

Figure 96. Global Clock Control Block



Related Links

Clock Control Block (ALTCLKCTRL) Megafunction User Guide

8.2.4 Instantiate Partial Reconfiguration Control Block in the Design

When you instantiate the Arria 10 Partial Reconfiguration Controller IP core in your design, the Quartus Prime software automatically connects the PR control block with the PR Controller IP core. However, if you are writing your own custom logic to perform the function of the Partial Reconfiguration Controller IP core, manually instantiate the control block to communicate with the FPGA system.

The Partial Reconfiguration Controller IP core interfaces with the PR control block to manage the bitstream source. Use this IP core in your design when performing partial reconfiguration using an internal PR host, Nios II, PCI Express*, or Ethernet. Instantiate the IP core either from the Quartus Prime IP catalog, or using Qsys Pro.

During partial reconfiguration, send a PR bitstream stored outside the FPGA to the PR control block inside the FPGA. This communication enables the control block to update the CRAM bits necessary for configuring the PR region in the FPGA. The PR bitstream contains the instructions (opcodes) and the configuration bits necessary for reconfiguring a specific PR region.

Related Links

Partial Reconfiguration IP Solutions User Guide

8.2.4.1 Component Declaration of the PR Control Block and CRC Block in VHDL

To manually instantiate the PR control block and the CRC block in your design:

1. Use the code sample below, containing the component declaration in VHDL. This code performs the PR function from within the core (code block within `Core_Top`).

```
-- The Arria 10 control block interface
component twentynm_prblock is
 port(
 corectl: in STD_LOGIC ;
 prrequest: in STD_LOGIC ;
 data: in STD_LOGIC_VECTOR(31 downto 0);
 error: out STD_LOGIC ;
 ready: out STD_LOGIC ;
 done: out STD_LOGIC
 );
end component;
-- The Arria 10 CRC block for diagnosing CRC errors

component twentynm_crcblock is
 port(
 shiftnld: in STD_LOGIC ;
 clk: in STD_LOGIC ;
 crcerror: out STD_LOGIC
 );
end component;
```

   Note: This VHDL example is adaptable for Verilog HDL instantiation.

2. Add additional ports to `Core_Top` to connect to both components.
3. Follow these rules when connecting the PR control block to the rest of your design:
   - Set the `corectl` signal to '1' (when using partial reconfiguration from core) or to '0' (when using partial reconfiguration from pins).
   - The `corectl` signal must match the **Enable PR pins** option setting in the **Device and Pin Options** dialog box on the **Settings** page (**Assignments ➤ Settings**).
   - When performing partial reconfiguration from pins, the Fitter automatically assigns the PR unassigned pins. Assign all the dedicated PR pins using Pin Planner (**Assignments ➤ Pin Planner**) or Assignment Editor (**Assignments ➤ Assignment Editor**).
   - When performing partial reconfiguration from the core logic, connect the `prblock` signals to either core logic or I/O pins, excluding the dedicated programming pin, such as DCLK.

8.2.4.1.1 Instantiating the PR Control Block and CRC Block in VHDL

The following example instantiates a PR control block inside your top-level project, Chip_Top, in VHDL:

```
module Chip_Top is port (
 --User I/O signals (excluding PR related signals)
 ..
 ..
 )
-- Following shows the connectivity within the Chip_Top module
Core_Top : Core_Top
 port_map (
 ..
 ..
 );
m_pr : twentynm_prblock
 port map(
clk => dclk,
 corectl =>'1', --1 - when using PR from inside
 --0 - for PR from pins; You must also enable
 -- the appropriate option in Quartus Prime settings
 prrequest => pr_request,
 data => pr_data,
 error => pr_error,
 ready => pr_ready,
 done => pr_done
 );
m_crc : twentynm_crcblock
 port map(
 shiftnld => '1', --If you want to read the EMR register when
 clk => dummy_clk, --error occurrs, refer to AN539 for the
 --connectivity forthis signal. If you only want
 --to detect CRC errors, but plan to take no
 --further action, you can tie the shiftnld
 --signal to logical high.
 crcerror => crc_error
 );
```

8.2.4.1.2 Instantiating the PR Control Block and CRC Block in Verilog HDL

The following example instantiates a PR control block inside your top-level project, Chip_Top, in Verilog HDL:

```
Chip_Top:
module Chip_Top (
 //User I/O signals (excluding PR related signals)
..
..
//PR interface and configuration signals declaration
 wire pr_request;
 wire pr_ready;
 wire pr_done;
 wire crc_error;
 wire dclk;
 wire [31:0] pr_data;

twentynm_prblock m_pr
 (
 .clk (dclk),
 .corectl (1'b1),
 .prrequest(pr_request),
 .data (pr_data),
 .error (pr_error),
 .ready (pr_ready),
 .done (pr_done)
```

```
 );

twentynm_crcblock m_crc
 (
 .clk (clk),
 .shiftnld (1'b1),
 .crcerror (crc_error)
 );
endmodule
```

For more information on port connectivity for reading the Error Message Register (EMR), refer to the *AN539: Test Methodology of Error Detection and Recovery using CRC in Altera FPGA Devices* application note.

Related Links

AN539: Test Methodology of Error Detection and Recovery using CRC in Altera FPGA Devices

8.2.4.2 Partial Reconfiguration Control Block Signals

The following table lists the partial reconfiguration control block interface signals:

Table 70. PR Control Block Interface Signals

Signal	Width	Direction	Description
data	[31:0]	Input	Carries the configuration bitstream.
done	1	Output	Indicates that the PR process is complete.
ready	1	Output	Indicates that the control block is ready to accept PR data from the control logic.
error	1	Output	Indicates a partial reconfiguration error.
prrequest	1	Input	Indicates that the PR process is ready to begin.
corectl	1	Input	Determines whether you are performing the partial reconfiguration internally, or through pins.

Note:

- Use `data` signal width of x8, x16, or x32 in your PR design.
- All the inputs and outputs are asynchronous to the PR clock (`clk`), except `data` signal. `data` signal is synchronous to `clk` signal.
- PR clock must be free-running.
- `data` signal must be 0 while waiting for `ready` signal.

8.2.4.2.1 PR Control Block Signals Timing Diagrams

Successful PR Session

The following flow describes a successful PR session:

1. Assert PR_REQUEST and wait for PR_READY; drive PR_DATA to 0.

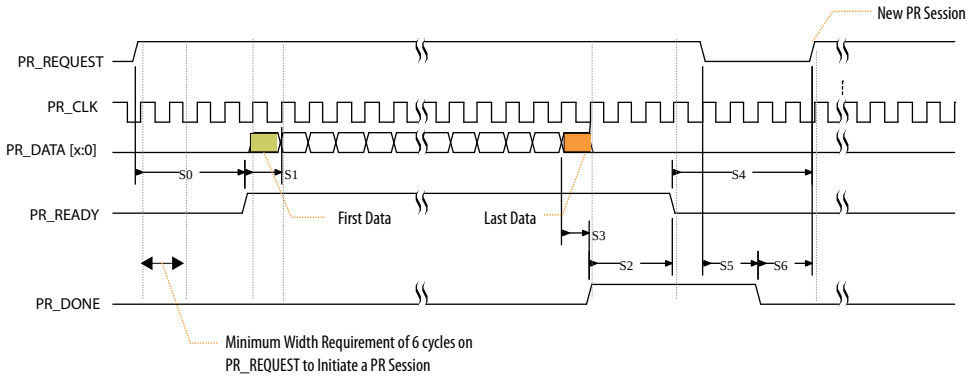
The PR control block asserts PR_READY, asynchronous to `clk`.

2. Start sending Raw Binary File (RBF) to PR control block, with 1 valid word per clock cycle. On `.rbf` file transfer completion, drive PR_DATA to 0. For more information on the `.rbf` file format, refer to *Raw Binary Programming File Format*. The PR control block asynchronously asserts PR_DONE when the control block completes the reconfiguration operation. PR control block deasserts PR_READY on configuration completion.
3. Deassert PR_REQUEST.

   The PR control block acknowledges the end of PR_REQUEST, and deasserts PR_DONE.

   The host can now initiate another PR session.

Figure 97. Timing Diagram for Successful PR Session



Related Links

Raw Binary Programming File Format on page 288

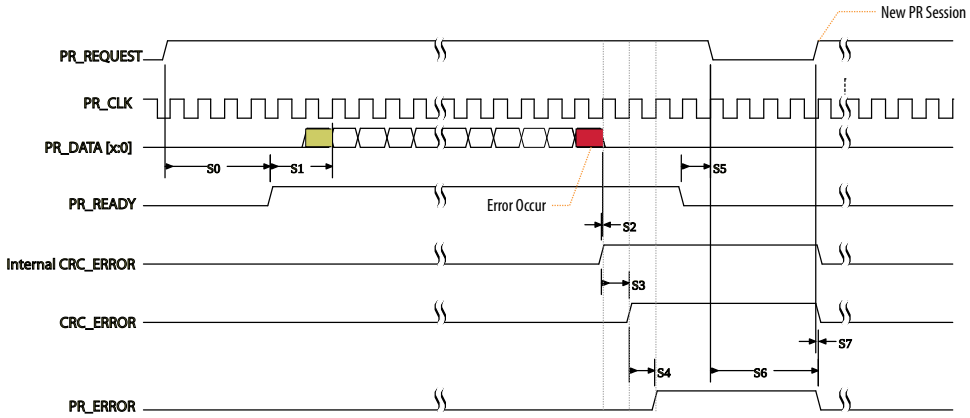
Unsuccessful PR Session with Configuration Frame Readback Error

The following flow describes a PR session with error in the EDCRC verification of a configuration frame readback:

1. The PR control block internally detects a CRC error.
2. The CRC control block then asserts CRC_ERROR.
3. The PR control block asserts the PR_ERROR.
4. The PR control block deasserts PR_READY, so that the host can withdraw the PR_REQUEST.
5. The PR control block deasserts CRC_ERROR and clears the internal CRC_ERROR signal to get ready for a new PR session.

The host can now initiate another PR session.

Figure 98. Timing Diagram for Unsuccessful PR Session with Configuration Frame Readback Error



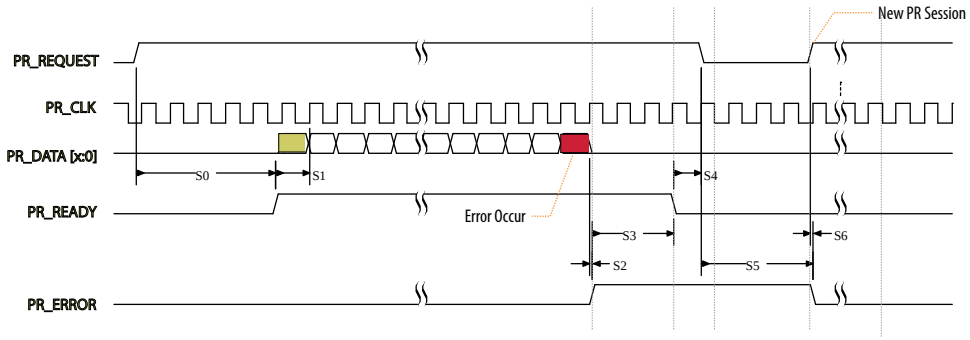
Unsuccessful PR Session with PR_ERROR

The following flow describes a PR session with transmission error or configuration CRC error:

1. The PR control block asserts PR_ERROR.
2. The PR control block deasserts PR_READY, so that the host can withdraw PR_REQUEST.
3. The PR control block deasserts PR_ERROR to get ready for a new PR session.

The host can now initiate another PR session.

Figure 99. Timing Diagram for Unsuccessful PR Session with PR_ERROR



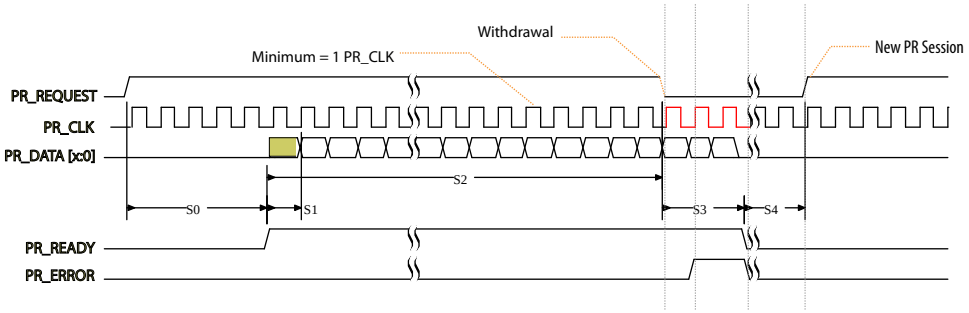
Late Withdrawal PR Session

The following flow describes a late withdrawal PR session:

1. The PR host can withdraw the request after the PR control block asserts PR_READY.
2. The PR control block deasserts PR_READY.

The host can now initiate another PR session.

Figure 100. Timing Diagram for Late Withdrawal PR Session

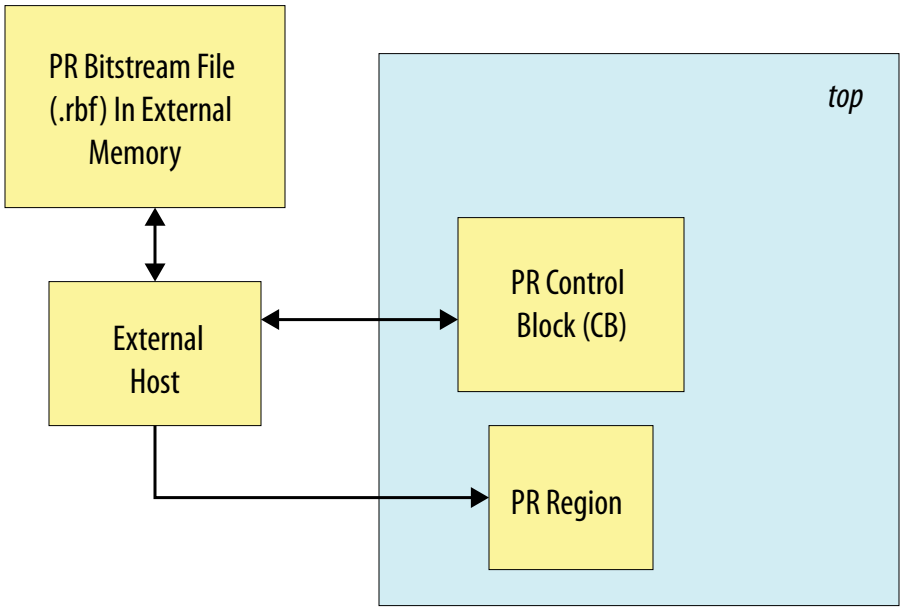


Note: The PR host can withdraw the request any time before the controller asserts PR_READY. Therefore, the PR host must not return until the PR control block asserts PR_READY. Provide at least 10 PR_CLK cycles after deassertion of PR_REQUEST, before requesting a new PR session.

8.2.4.3 External Host PR

In external host control, an external FPGA or CPU controls the PR configuration using external dedicated PR pins on the target device. When using an external host, implement the control logic for managing the partial reconfiguration system on an external device. This implementation includes correct freezing of all the PR region outputs, and resetting the freeze signal on successful PR completion.

Figure 101. PR System Using an External Host



To use an external host for your design:

1. Click **Assignments ➤ Device ➤ Device and Pin Options**.
2. Select the **Enable PR Pins** option in the **Device and Pin Options** dialog box. This option automatically creates the special partial reconfiguration pins, and defines the pins in the device pin-out. This option also automatically connects the pins to PR control block internal path.

   Note: If you do not select this option, you must use an internal or HPS host. You do not need to define pins in your design top-level entity.
3. Wire these top-level pins to the specific ports in the PR control block.

The following table lists the automatically constrained PR pins when you select **Enable PR Pins** option, and the specific PR control block port to connect these pins to:

Table 71. Partial Reconfiguration Dedicated Pins

Pin Name	Type	PR Control Block Port Name	Description
PR_REQUEST	Input	prrequest	Logic high on this pin indicates that the PR host is requesting partial reconfiguration.
PR_READY	Output	ready	Logic high on this pin indicates that the PR control block is ready to begin partial reconfiguration.
PR_DONE	Output	done	Logic high on this pin indicates that the partial reconfiguration is complete.
PR_ERROR	Output	error	Logic high on this pin indicates an error in the device during partial reconfiguration.
DATA[31:0]	Input	data	These pins provide connectivity for PR_DATA to transfer the PR bitstream to the PR controller.
DCLK	Input	clk	Receives synchronous PR_DATA.

Note:

1. PR_DATA can be 8, 16, or 32-bits in width.
2. Ensure that you connect the `corectl` port of the PR control block to 0.

During user mode, the external host initiates partial reconfiguration and monitors the status using the external PR dedicated pins. In this mode, the external host must respond appropriately to the handshake signals for successful partial reconfiguration. The external host writes the partial bitstream data from external memory into the Arria 10 device. Co-ordinate system-level partial reconfiguration by ensuring that you prepare the correct PR region for partial reconfiguration. After reconfiguration, return the PR region into operating state.

Example 71. Verilog RTL for External Host PR

```
module top(
     // PR control block signals
     input  logic        pr_clk,
     input  logic        pr_request,
```

```
     input  logic [31:0] pr_data,
     output logic        pr_error,
     output logic        pr_ready,
     output logic        pr_done,

     // User signals
     input  logic i1_main,
     input  logic i2_main,
     output logic o1
);

// Instantiate the PR control block
twentynm_prblock m_prblock
(
    .clk(pr_clk),
    .corectl(1'b0),
    .prrequest(pr_request),
    .data(pr_data),
    .error(pr_error),
    .ready(pr_ready),
    .done(pr_done)
);


// PR Interface partition
pr_v1 pr_inst(
   .i1(i1_main),
   .i2(i2_main),
   .o1(o1)
);

endmodule
```

Example 72. VHDL RTL for External Host PR

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

entity top is
port(
      -- PR control block signals
      pr_clk: in std_logic;
      pr_request: in std_logic;
      pr_data: in std_logic_vector(31 downto 0);

      pr_error: out std_logic;
      pr_ready: out std_logic;
      pr_done: out std_logic;

      -- User signals
      i1_main: in std_logic;
      i2_main: in std_logic;
      o1: out std_logic
);
end top;

architecture behav of top is

component twentynm_prblock is
port(
      clk: in std_logic;
      corectl: in std_logic;
      prrequest: in std_logic;
      data: in std_logic_vector(31 downto 0);
      error: out std_logic;
      ready: out std_logic;
      done: out std_logic
);
end component;
```

```
component pr_v1 is
port(
      i1: in std_logic;
      i2: in std_logic;
      o1: out std_logic
);
end component;

signal pr_gnd : std_logic;

begin

pr_gnd <= '0';

-- Instantiate the PR control block
m_prblock: twentynm_prblock port map
(
    pr_clk,
    pr_gnd,
    pr_request,
    pr_data,
    pr_error,
    pr_ready,
    pr_done
);


-- PR Interface partition
pr_inst : pr_v1 port map
(
   i1_main,
   i2_main,
   o1
);

end behav;
```

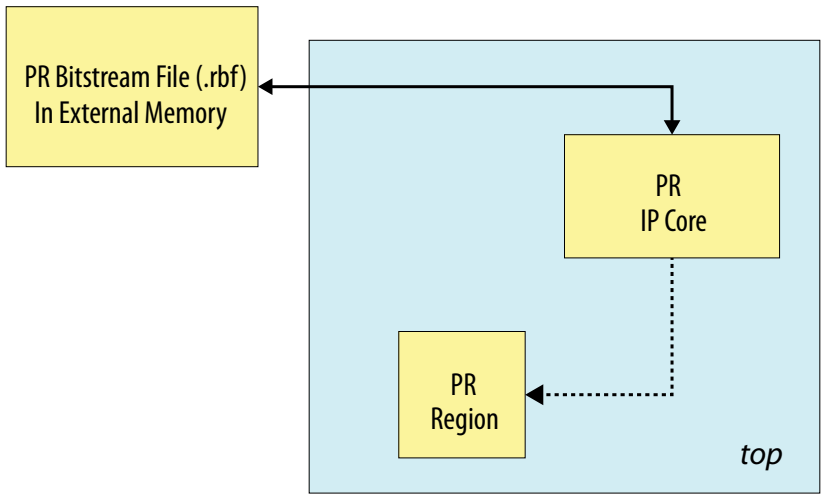
8.2.4.4 Internal Host PR

In internal host control, an internal controller, a Nios II processor, or an interface such as PCI Express (PCIe) communicates directly with the PR control block instantiation.

To transfer the PR bitstream into the PR control block, use the Avalon-MM interface on the Partial Reconfiguration IP core. The external interfaces include PCIe, or controllers, such as the Nios II processor. Use the Partial Reconfiguration IP core to instantiate the PR control block. When the device enters user mode, initiate partial reconfiguration through the FPGA core fabric using the PR internal host.

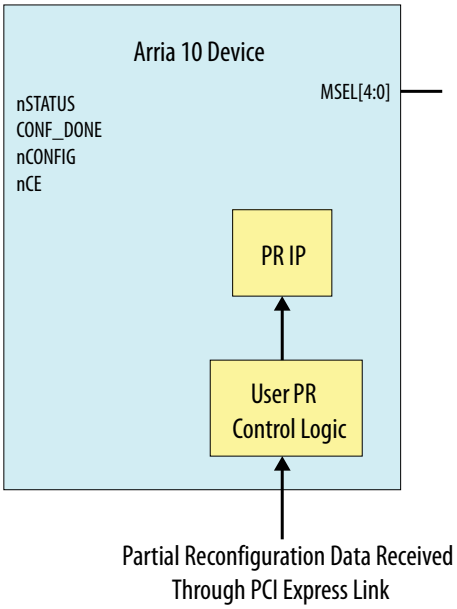
Note: If you create your own control logic for the PR host, you must meet the PR interface requirements.

Figure 102. Internal Host PR



When performing partial reconfiguration with an internal host, use the dedicated PR pins (PR_REQUEST, PR_READY, PR_DONE, and PR_ERROR) as regular I/Os. Implement your static region logic to retrieve the PR programming bitstreams from an external memory, for processing by the internal host.

Example 73. FPGA System Using an Internal PR Host



Receive the programming bitstreams for partial reconfiguration through the PCI Express link. Then, you process the bitstreams with your PR control logic and send the bitstreams to the Partial Reconfiguration IP core for programming.

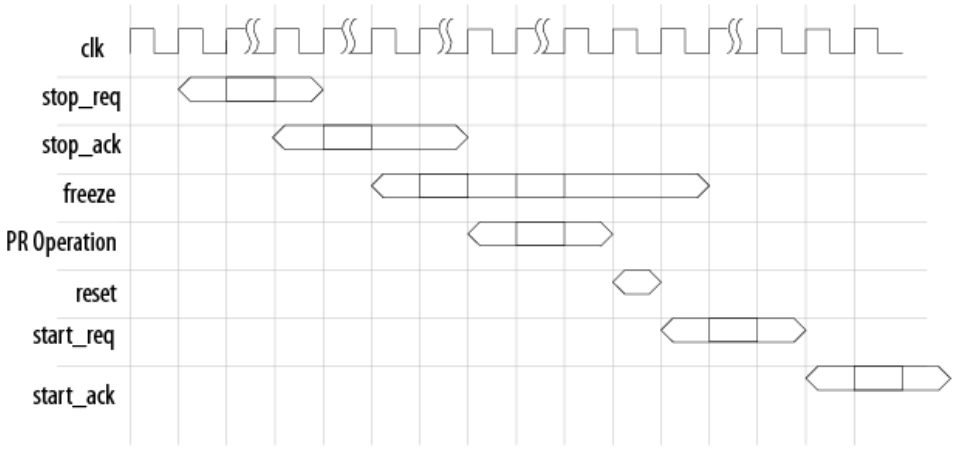
8.2.4.5 Partial Reconfiguration Process Sequence

The partial reconfiguration design initiates the PR operation, and delivers the configuration file to the PR control block as part of the system level design.

Before partial reconfiguration, ensure that the device is in user mode and a functional state. The following steps describe the PR sequence:

1. Send a `stop_req` signal to the PR region from the sequential PR control logic to prepare for the PR operation. This signal informs the PR regions to complete any pending transactions and stop accepting new transactions.
2. Wait for the `stop_ack` signal to indicate that the PR region is ready for partial reconfiguration.
3. Use PR control logic to freeze all necessary outputs of the PR regions. Additionally, drive the clock enable for any initialized RAMs to disabled state.
4. To initiate the PR process for the PR region, send the PR bitstream to the PR control block. When using the Partial Reconfiguration IP core in the design, the Avalon-MM or Avalon-ST interface on the IP core handles the process. When directly instantiating the PR control block, follow the PR control block interface protocol timings to ensure that the PR process progresses correctly.
5. On successful completion of the PR operation, reset the PR region.
6. Signal the PR region to start operating by asserting the `start_req` signal, and deasserting the `freeze` signal.
7. Wait for the `start_ack` signal to indicate that the PR region is ready for operation.
8. Resume operation of the FPGA with the newly configured PR region.

Figure 103. Recommended Process Sequence Timing Diagram



8.2.4.6 Reset the PR Region Registers

Upon partial reconfiguration of a PR region, the status of the PR region registers become indeterminate. Bring the registers in the PR region to a known state by applying a reset sequence for the PR region. This reset ensures that the system behaves to your specifications. Simply reset the control path of the PR region, if the datapath is eventually flushed out within a finite number of cycles.

Table 72. Supported PR Reset Implementation Guideline

PR Reset Type	Active-High Synchronous Reset	Active-High Asynchronous Reset	Active-Low Synchronous Reset	Active-Low Asynchronous Reset
On local signal	Yes	Yes	Yes	Yes
On global signal	No	Yes	No	Yes

Note: Use active-high local reset instead of active-low, wherever applicable. This action allows you to automatically hold the PR region in reset, by virtue of the boundary port wire LUT.

8.2.4.7 Promote Global Signals in a PR Region

In standard designs, the Quartus Prime software automatically promotes high fan-out signals onto dedicated global networks. This global promotion happens during the planning stage of design compilation.

In PR designs, the Compiler disables global promotion for signals originating within the logic of a PR region. Instantiate the clock control blocks only in the static region, because the clock floorplan and the clock buffers must be a part of the static region of the design. Manually instantiating a clock control block in a PR region, or assigning a signal in a PR region with the `GLOBAL_SIGNAL` assignment results in compilation error. To drive a signal originating from the PR region onto a global network:

1. Export the signal from the PR region.
2. Drive the signal onto the global network from the static region.
3. Drive the signal back into the PR region.

For Arria 10 devices, you can drive a maximum of 33 clocks into any PR region, but you cannot share a row clock between two PR regions. Use the Chip Planner to visualize the row clock region boundaries, and to ensure that no two PR regions share a row clock region.

When promoting global signals, the Compiler allows only certain signals to be global inside the PR regions. Use only global signals to route certain secondary signals into a PR region. The following table lists the restriction for each block:

Table 73. Supported Signal Types for Driving Clock Networks in a PR Region

Block Type	Supported Global Network Signals
LAB, MLAB	Clock, ACLR
RAM, ROM (M20K)	Clock, ACLR, Write Enable (WE), Read Enable (RE)
DSP	Clock, ACLR

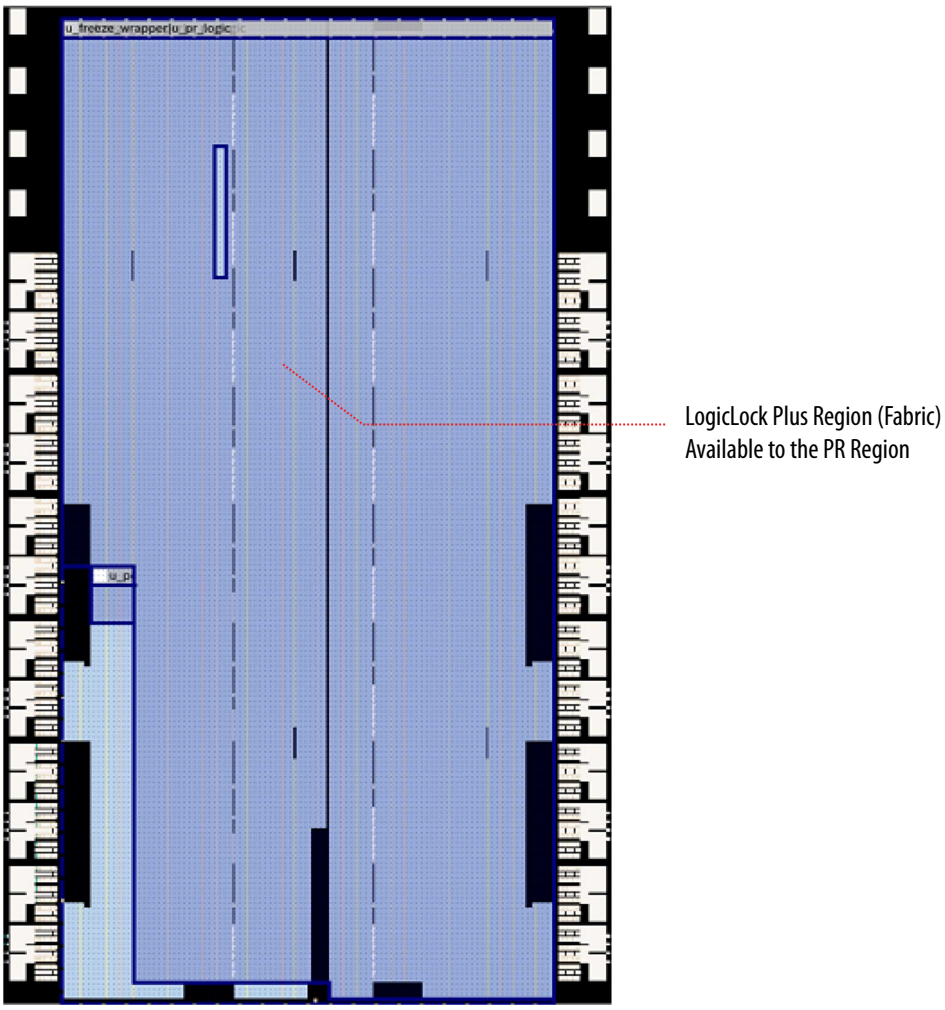
8.2.5 Floorplan the Partial Reconfiguration Design

The floorplan constraints in your partial reconfiguration design physically partitions the device. This partitioning ensures that the resources available to the PR region are the same for any persona that you implement.

Your design must include only core logic, such as LABs, RAMs, ROMs, and DSPs in a PR region. Instantiate all periphery design elements, such as transceivers, external memory interfaces, and clock networks in the static region of the design. However, the

LogicLock Plus regions can cross periphery locations, such as the I/O columns and the HPS, because the constraint is core-only. The following figure shows the PR region floorplan covering the I/O columns in the middle of the device:

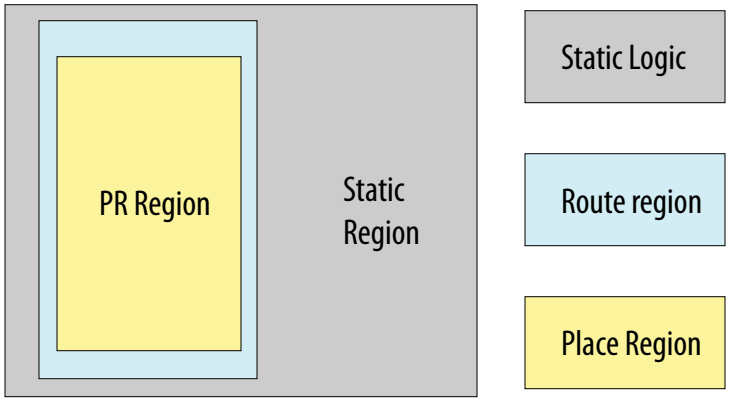
Figure 104. PR Region Floorplan



To create periphery floorplan assignments for your design, use the BluePrint Platform Designer.

Note: Complete the periphery and clock floorplan before core floorplanning.

Figure 105. Floorplanning your PR Design



Each PR partition in your design must have a corresponding, exclusive physical partition. Assign the LogicLock Plus region(s) to define the physical partition for your PR region. There are two region types:

- Placement regions—use these regions to constrain logic to a specific area of the device. The Fitter places the logic in the region you specify. The Fitter can also place other logic in the region unless you designate the region as **Reserved**.
- Routing regions—use these regions to constrain routing to a specific area.

The routing region must fully enclose the placement region. Additionally, the routing regions for the PR regions cannot overlap.

Create LogicLock Plus regions from the Project Navigator, LogicLock Plus Regions window, or Chip Planner. For complete information on creating LogicLock Plus regions, refer to *Creating LogicLock Plus Regions* section in Volume 2 of the Quartus Prime Pro Edition Handbook.

Follow these guidelines when floorplanning your PR design:

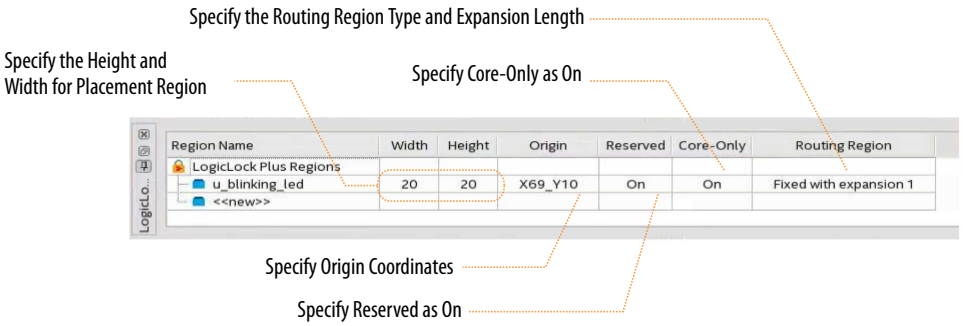
- Define a routing region that is at least 1 unit larger than the placement region in all directions.
- Do not overlap the routing regions of multiple PR regions.
- Select the PR region row-wise for least bitstream overhead. In Arria 10 devices, the short, wide regions have smaller bitstream size compared to tall, narrow regions.
- Define sub LogicLock Plus regions within PR regions to improve timing closure.
- The height of your floorplan affects the reconfiguration time. A floorplan larger in the Y direction takes longer to reconfigure.
- If your design includes a hierarchical PR flow with parent and child partitions, placement region of the parent region must fully enclose the routing and placement region of its child region. Also, the parent wire LUTs must be in an area, outside the child PR region. This requirement is because the child PR region is exclusive to all other logic, which includes the parent and the static region.

To create a LogicLock Plus region for your PR partition:

1. Right-click the design instance in the **Project Navigator** and click **LogicLock Plus Region ➤ Create New LogicLock Plus Region**. The region appears on the LogicLock Plus Regions Window.
2. Specify the placement region co-ordinates in the **Origin** column. The origin corresponds to the lower-left corner of the region. For example, to set a place region with (X1 Y1) co-ordinates as (69 10), specify the **Origin** as X69_Y10. The Quartus Prime software automatically calculates the (X2 Y2) co-ordinates (top-right) for the place region, based on the height and width you specify.
3. Enable the **Reserved** and **Core-Only** options.
4. Double-click the **Routing Region** option. The **LogicLock Plus Routing Region Settings** dialog box appears.
5. Specify the **Routing type**. The LogicLock Plus region supports the following routing types:
   - **Whole chip**—allocates the entire chip for the routing shape.
   - **Fixed with expansion**—allocates an expansion length of 1 for the routing shape.
   - **Custom**—allows you to manually add a custom routing shape and specify the **Height**, **Width**, and **Origin**.

     Note: The routing shape must be larger than the placement shape.
6. Click **OK**.

Figure 106. LogicLock Plus Regions Window



The following assignments in the `.qsf` file correspond to creating a core-only, reserved LogicLock Plus region with placement and routing regions:

```
set_instance_assignment -name PLACE_REGION "69 10 88 29" -to <design_instance>
set_instance_assignment -name RESERVE_PLACE_REGION ON -to <design_instance>
set_instance_assignment -name CORE_ONLY_PLACE_REGION ON -to <design_instance>
set_instance_assignment -name ROUTE_REGION "68 9 89 30" -to <design_instance>
```

Related Links

- LogicLock Plus Regions
  For complete information on how to create LogicLock Plus regions.
- BluePrint Design Planning
  For complete information on BluePrint Platform Designer.

8.2.5.1 Incrementally Implementing Partial Reconfiguration

Successfully implementing a partial reconfiguration design requires several additional constraints for identifying the reconfigurable parts of the design and device. As these constraints significantly impact the timing closure ability of the Compiler, incrementally implement the required constraints and analyze each result.

Note: PR designs require a more constrained floorplan, compared to a flat design. Hence, the overall density and performance of a PR design may be lower than an equivalent flat design.

The following steps describe incrementally developing the requirements for your PR design:

1. Implement the base revision using the most complex persona for each PR partition. This initial implementation must include the complete design with all periphery constraints and top-level `.sdc` timing constraints. Do not include any LogicLock Plus region constraints for the PR regions with this implementation.
2. Create partitions by disabling the **Reconfigurable** option in the Design Partitions Window, for all the PR partitions.
3. Register the boundaries of each partition to ensure adequate timing margin.
4. Verify successful timing closure using the TimeQuest Timing Analyzer.
5. Ensure that all the desired signals are driven on global networks. Disable the **Auto Global Clock** option in the Fitter (**Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Fitter)**), to avoid promoting non-global signals.
6. Create LogicLock Plus core-only placement regions for each of the partitions.
7. Recompile the base revision with these new constraints and verify timing closure.
8. Enable the **Reserved** option for each LogicLock Plus region to ensure the exclusive placement of the PR partitions within the placement regions.

   Note: Enabling the **Reserved** option avoids placing the static region logic in the placement region of the PR partition.
9. Recompile the base revision with this new constraint and verify timing closure.
10. Create LogicLock Plus routing regions for each of the PR partitions. The routing region directs the Compiler to confine the routing for the instance within the defined region. The routing region must completely surround the associated placement region. Routing regions for PR partitions must be completely disjoint, and cannot overlap.

    Note: Routing regions are not exclusive, and other partitions, such as the top-level partition can route through this area.
11. Recompile the base revision with this new constraint and verify timing closure.
12. In the Design Partitions Window, specify each of the PR partitions as **Reconfigurable**. This assignment ensures that the Compiler adds wire LUTs for each interface of the PR partition, and performs additional compilation checks for partial reconfiguration.
13. Recompile the base revision with this new constraint and verify timing closure.

You can now export the top-level partition for reuse in the PR implementation compilation of the different personas.

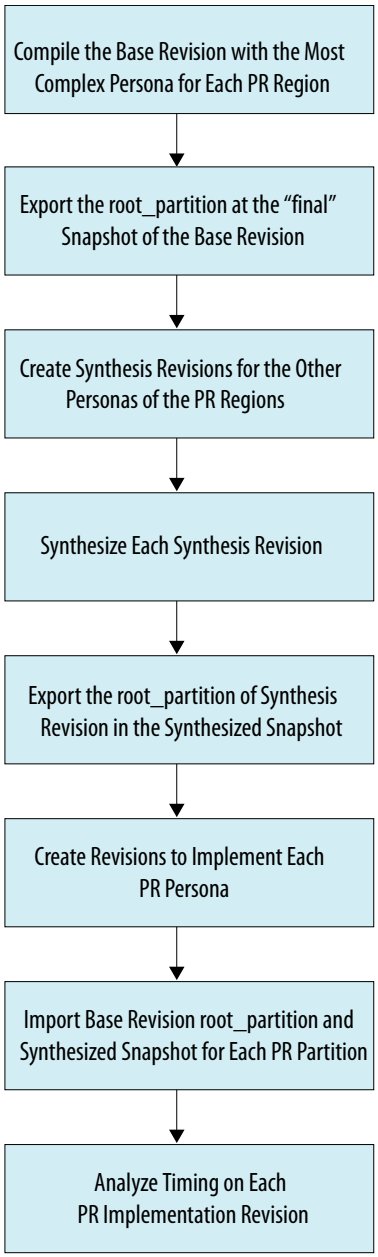
Related Links

Create Design Partitions for Partial Reconfiguration on page 249
For more information on creating design partitions for partial reconfiguration.

8.2.6 Create Revisions for Personas

To compile a partial reconfiguration project, create a base revision for the design. Also, create synthesis and PR implementation revisions for each of the personas.

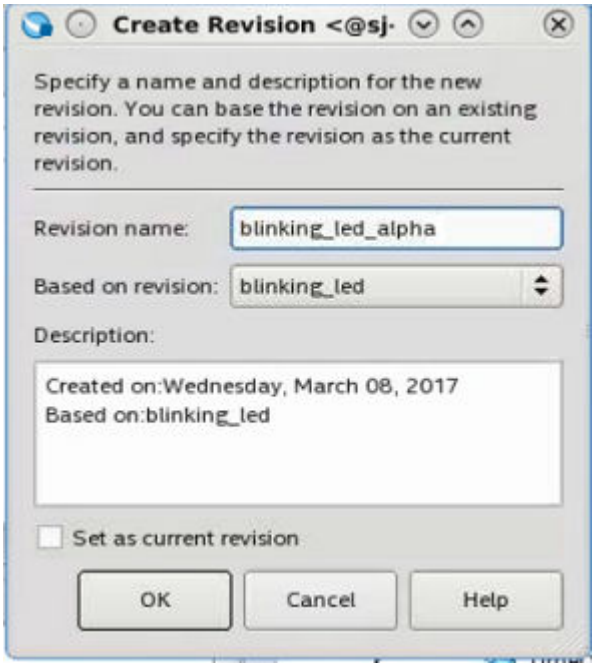
Figure 107. Partial Reconfiguration Compilation Flow for Arria 10 Devices



To create the PR implementation revisions:

1. To open the Revisions window, click **Project ➤ Revisions**.
2. To create a new revision, double-click **<<new revision>>**.
3. Specify the **Revision name**.
4. Select the **Based on revision** option.
5. Enable **Set as current revision** to specify the persona as your current revision, and click **OK**.

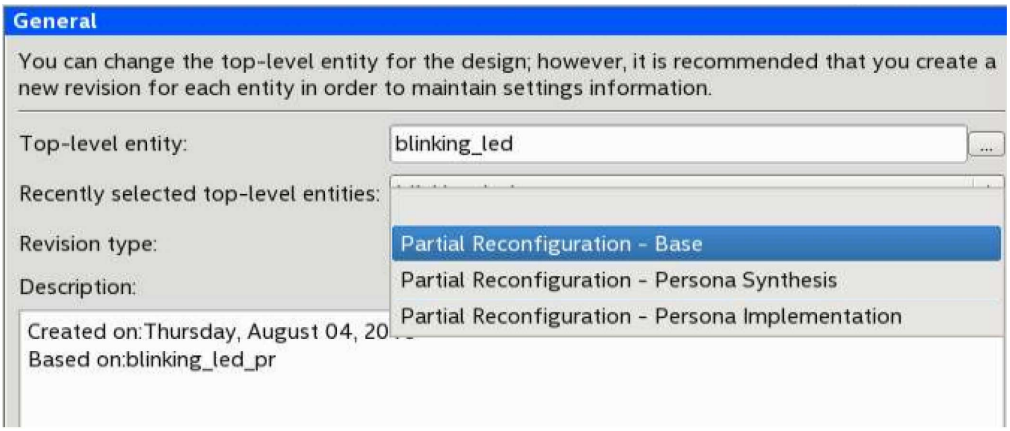
Figure 108. Create Revisions



To set the revision type:

1. Click **Assignments ➤ Settings**.
2. Click the **General** tab.
3. Select the persona for setting the revision type from the **Recently selected top-level entities** list.
4. Select the revision type:
   - **Partial Reconfiguration - Base**
   - **Partial Reconfiguration - Persona Synthesis**
   - **Partial Reconfiguration - Persona Implementation**
5. Click **Apply** and **OK**.

Figure 109. Specify Revision Type



The following assignments in the respective revision's `.qsf` file correspond to specifying the revision type from the **Settings** dialog box:

```
set_global_assignment -name REVISION_TYPE PR_BASE
set_global_assignment -name REVISION_TYPE PR_SYN
set_global_assignment -name REVISION_TYPE PR_IMPL
```

8.2.7 Compile the Partial Reconfiguration Design

The number of compilations a PR design requires depends on the number of PR personas. Use the base revision compilation, and each PR implementation compilation for timing analysis.

Typically, you compile a partial reconfiguration design in two phases:

1. Place and route the static partitions, along with a set of default personas for each PR partition.
2. Compile the alternate personas, while preserving the static partition's placing and routing blocks.

When reusing or preserving a design block, always specify the precise compilation snapshot to reuse. For example, when compiling the alternate personas of a PR design, specify the snapshot for that compilation as the "final" snapshot of the static region. Otherwise, the Compiler cannot preserve the routing information.

Related Links

Design Compilation
For more information on how to analyze, synthesize, place, and route your design.

8.2.7.1 Using the Partial Reconfiguration Flow Script

The Quartus Prime Pro Edition software provides a flow template for compiling a partial reconfiguration design in Arria 10 devices.

To create the template, `a10_partial_reconfig/flow.tcl` in your Quartus Prime project directory, type the following command from the Quartus Prime shell:

```
quartus_sh --write_flow_template –flow a10_partial_reconfig
```

To run this script from the Quartus Prime shell, type the following command:

```
quartus_sh –t a10_partial_reconfig/flow.tcl
```

Use the following options when running this script:

Table 74. Partial Reconfiguration Flow Script Options

Option	Description
`-all`	Default option that compiles the base revision and all the PR implementation revisions.
`--impl[=<name>]`	Compiles a specified PR implementation. Specify the revision name of the implementation to compile.
`-all_impl`	Compiles all PR implementations. Skips the base revision compilation.
`-base`	Compiles the base revision. Skips all the PR implementations compilation.
`-check`	Checks the script configuration and exits without performing any compilation.
`-setup_script[=<file_name>]`	Allows you to customize the script settings with your partial reconfiguration project details. The settings you define in this file override the variable settings in `a10_partial_reconfig/setup.tcl` template.

8.2.7.1.1 Configuring the Partial Reconfiguration Flow Script

To configure the PR flow script for your design:

1. Rename the generated `a10_partial_reconfig/setup.tcl.example` to `a10_partial_reconfig/setup.tcl`.
2. Edit the `setup.tcl` file with configuration that overrides the variable settings in the `a10_partial_reconfig/flow.tcl` file. To define the name of your Quartus Prime partial reconfiguration project, modify the following line:

   ```
   define_project <project_name>
   ```

   Note: All revisions must be present in the corresponding `.qpf` file.
3. To define the base revision name, modify the following line:

   ```
   define_base_revision <base_revision_name>
   ```

This revision represents the static region of the design.

4. To define each of the partial reconfiguration implementation revisions, along with the PR partition names and the synthesis revision that implements the revisions, modify the following line:

```
define_pr_impl_partition -impl_rev_name <implementation_revision_name> \
    -partition_name <pr_partition_name\
    -source_rev_name <synthesis_revision_name>
...
...
```

Note: Alternatively, use the `setup_script` option while running the `flow.tcl` script to specify the `setup.tcl` configuration file location.

8.2.7.1.2 Running the Partial Reconfiguration Flow Script

To run the partial reconfiguration flow script with your setup file:

1. Click **Tools ➤ Tcl Scripts**. The **Tcl Scripts** dialog box appears.
2. Click **Add to Project**, browse and select the `a10_partial_reconfig/flow.tcl`.
3. Select the `a10_partial_reconfig/flow.tcl` in the Libraries pane, and click **Run**.

Alternatively, to run the script from the Quartus Prime command shell, type the following command:

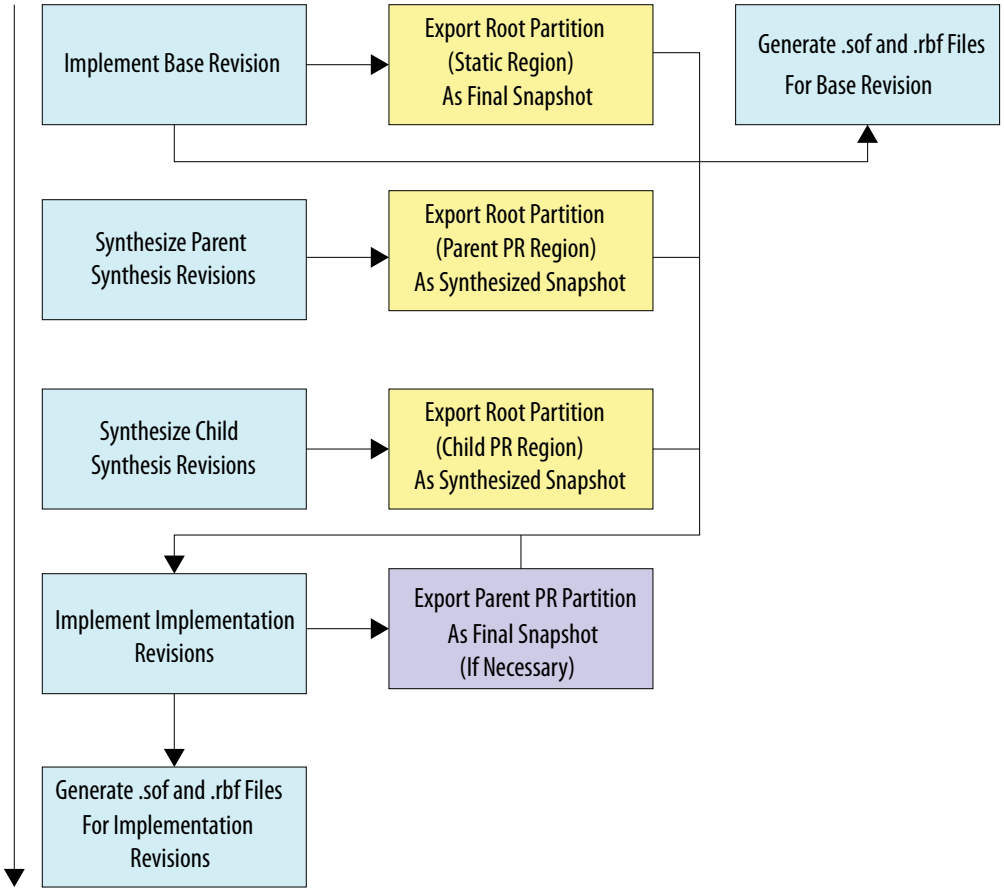
```
quartus_sh -t a10_partial_reconfig/flow.tcl -setup_script setup.tcl
```

8.2.7.2 Hierarchical Partial Reconfiguration Compilation Flow

Similar to compiling a regular PR design, you must create a base revision for your design. Create dedicated synthesis revisions for each parent and child partition in your design. Specify the design file of the parent and child partition as the top-level entity for the corresponding synthesis revision. Also, ensure that the synthesis revisions for the parent PR partitions include the partition assignments for its child PR regions.

Unlike the implementation revisions in a regular PR design, the hierarchical PR implementation revisions require a complete floorplan, as well as PR partition assignments. The reasoning behind this requirement is the possible change in the PR partition assignment and floorplan for the child partitions, between different implementation revisions. Also, if you are implementing the same parent PR persona as the final snapshot in multiple implementation revisions, ensure that the floorplan for this parent PR partition and its child partition(s) are the same across all implementation revisions.

Figure 110. Hierarchical Partial Reconfiguration Compilation Flow for Arria 10 Devices



8.2.7.2.1 Configuring the Hierarchical Partial Reconfiguration Flow Script

To configure the HPR flow script for your design:

1. Rename the generated `a10_hier_partial_reconfig/setup.tcl.example` to `a10_hier_partial_reconfig/setup.tcl`.
2. Edit the `setup.tcl` file with configuration that overrides the variable settings in the `a10_hier_partial_reconfig/flow.tcl` file. To define the name of your Quartus Prime hierarchical partial reconfiguration project, modify the following line:

```
define_project <project_name>
```

   Note: All revisions must be present in the corresponding `.qpf` file.
3. To define the base revision name, modify the following line:

```
define_base_revision <base_revision_name>
```

This revision represents the static region of the design.

4. To define each parent and child partition in each of the implementation revisions, along with the partition names, the implementation revision name, source revision name, source revision partition name, and the source snapshot, modify the following line:

```
define_pr_impl_partition -impl_rev_name <imple_revision_name> \
            -partition_name <partition_name> \
            -source_rev_name <source_revision_name> \
            -source_partition <source_partition_name> \
            -source_snapshot <source_snapshot>
```

Note: Alternatively, use the `setup_script` option while running the `flow.tcl` script to specify the `setup.tcl` configuration file location.

Table 75. Implementation Revision Definition Arguments

Argument	Description
`-impl_rev_name`	Defines the implementation revision name
`-partition_name`	Defines the partition name
`-source_rev_name`	Defines the name of the source revision. This revision can be a synthesis or implementation revision, from which this partition imports the exported synthesis/final snapshot, for implementation.
`-source_partition`	Defines the partition in the source revision, which the Compiler exports, later in the flow. This partition can either be a root partition for synthesis source revisions, or a parent PR partition for implementation source revisions.
`-source_snapshot`	Defines the snapshot of the source partition that the Compiler exports, later in the flow. Usually, you define this argument as the final snapshot for parent PR partitions exported from implementation revisions, and synthesized snapshot for root partitions exported from synthesis revisions.

Note: Alternatively, use the `setup_script` option while running the `flow.tcl` script to specify the `setup.tcl` configuration file location. For more information on how to configure the HPR flow script, refer to *Step 8: Generating the Hierarchical Partial Reconfiguration Flow Script* in the *AN 805: Hierarchical Partial Reconfiguration of a Design on Intel Arria 10 SoC Development Board* application note.

Related Links

Step 8: Generating the Hierarchical Partial Reconfiguration Flow Script

8.2.7.2.2 Running the Hierarchical Partial Reconfiguration Flow Script

To run the hierarchical partial reconfiguration flow script with your setup file:

1. Click **Tools ➤ Tcl Scripts**. The **Tcl Scripts** dialog box appears.
2. Click **Add to Project**, browse and select the `a10_hier_partial_reconfig/flow.tcl`.
3. Select the `a10_hier_partial_reconfig/flow.tcl` in the Libraries pane, and click **Run**.

Alternatively, to run the script from the Quartus Prime command shell, type the following command:

```
quartus_sh -t a10_hier_partial_reconfig/flow.tcl -setup_script
a10_hier_partial_reconfig/setup.tcl
```

Table 76. Hierarchical Partial Reconfiguration Flow Script Options

Option	Description
`-all`	Default option that compiles the base revision and all the PR implementation revisions.
`-all_syn`	Compiles all the HPR synthesis revisions. Skips the base revision compilation.
`-impl[=<name>]`	Compiles a specified HPR implementation revision. Specify the revision name of the implementation to compile.
`-all_impl`	Compiles all HPR implementations. Skips the base revision compilation.
`-base`	Compiles the base revision. Skips all the HPR implementations compilation.
`-check`	Checks the script configuration and exits without performing any compilation.
`-setup_script[=<file_name>]`	Allows you to customize the script settings with your partial reconfiguration project details. The settings you define in this file override the variable settings in `a10_hier_partial_reconfig/setup.tcl` template.

8.2.8 Run Timing Analysis for the Partial Reconfiguration Design

The interface between partial and static partitions remains the same for each PR implementation revision. Perform timing analysis on each PR implementation revision to ensure that there are no timing violations.

Run multiple timing analyses on each of the static and PR implementation revisions. Meet the different timing requirements for multiple PR personas by specifying different `.sdc` constraints for each persona. If you need timing constraints for the synthesis persona, include the constraints in the synthesis revisions. The target name must match the hierarchy of the persona at the top-level.

Note: LogicLock Plus regions impose placement constraints that affect the performance and resource utilization of your PR design. Ensure that the design has additional timing allowance and available device resources. Selecting the largest and most timing-critical persona as your base persona optimizes the timing closure.

Related Links

TimeQuest Timing Analyzer
For complete information on TimeQuest analyzer and timing analysis.

8.2.8.1 Run Timing Analysis on a Design with Multiple PR Partitions

To perform timing analysis on a design with multiple PR partitions, you must create an aggregate revision with all possible persona combination. This combination of personas from multiple reconfigurable revisions help create a complete design, suitable for timing analysis.

To create an aggregate revision and perform timing analysis on the aggregate revision:

1. To open the **Revisions** dialog box, click **Project ➤ Revisions**.
2. To create a new revision, double-click **<<new revision>>**.
3. Specify the **Revision name** and select the base revision for **Based on Revision**.
4. Ensure that you include all the `.sdc` and `.ip` files for the static and PR region.

   Note: To detect the clocks, the SDC file for the PR IP must follow any SDC that creates the clocks that the IP core uses. You facilitate this order by ensuring the `.ip` file for the PR IP core comes after any `.ip` files or SDC files used to create these clocks in the QSF file for your Quartus Prime project revision. For more information, refer to *Timing Constraints* section in the *Partial Reconfiguration IP Core User Guide*.
5. To export the post-fit database from the base compile (static partition), type the following command in the Quartus Prime shell:

```
quartus_cdb <base_revision> --export_block "root_partition" --snapshot final
--file "<base revision name>.qdb" --exclude_pr_subblocks
```

   Note: The static partition post-fit database is already available in the base revision. You can use this *<base revision name>*`.qdb` file from the base revision project folder, instead of regenerating the `.qdb` file using the above command.
6. To export the post-fit database from the multiple personas (PR implementation revisions), type the following commands in the Quartus Prime shell:

```
quartus_cdb <PR1 Fit revision> --export_block <PR1 Partition name> --snapshot
final --file "pr1.qdb"

quartus_cdb <PR2 Fit revision> --export_block <PR2 Partition name> --snapshot
final --file "pr2.qdb"
```

7. To import the post-fit databases of the static and PR region as aggregate revision, type the following commands in the Quartus Prime shell:

```
quartus_cdb <aggr_rev> --import_block "root_partition" --file "<base revision
name>.qdb"

quartus_cdb <aggr_rev> --import_block <PR1 partition name> --file "pr1.qdb"

quartus_cdb <aggr_rev> --import_block <PR2 Partition name> --file "pr2.qdb"
```

8. To integrate post-fit database of all the partitions, type the following command in the Quartus Prime shell:

```
quartus_fit <proj name> -c <aggr_rev>
```

   Note: The Fitter verifies the legality of the post-fit database, and combines the netlist for timing analysis. The Fitter does not reroute the design.
9. To perform timing analysis on the aggregate revision, type the following command in the Quartus Prime shell:

```
quartus_sta <proj name> -c <aggr_rev>
```

Run timing analysis on aggregate revision for all possible PR persona combination. If a specific persona fails timing closure, recompile the persona and perform timing analysis again.

Related Links

Partial Reconfiguration IP Core User Guide
For information on the timing constraints.

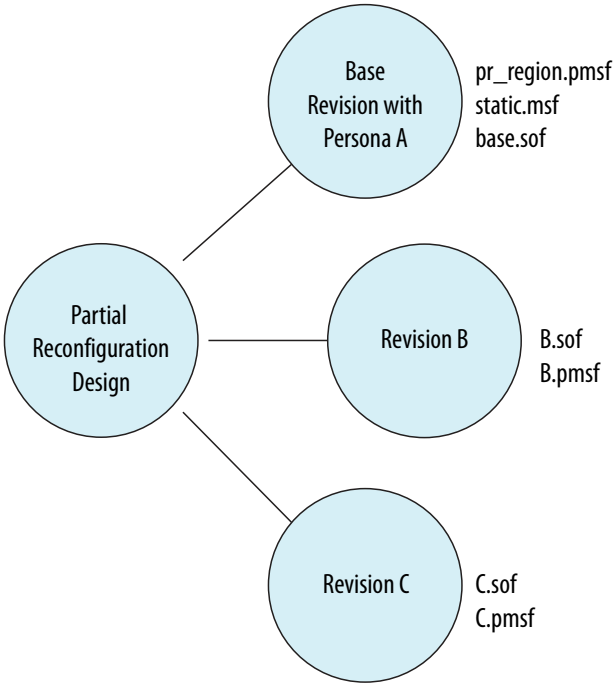
8.2.9 Generate Programming Files

You must generate partial reconfiguration bitstream(s) for the personas in your design. Send the bitstreams to the PR control block for partial reconfiguration. Compile the PR project, including the base revision and at least one reconfigurable revision before generating the PR bitstreams. The Quartus Prime Pro Edition Assembler generates the PR bitstreams. Send these generated bitstreams to the PR ports on the PR control block.

Example 74. Generated Programming Files for a Partial Reconfiguration Design

This example design contains a PR region and the following revisions:

- Base revision with persona A
- PR revision with persona B
- PR revision with persona C



When you compile these individual revisions, the Assembler produces Partial-Masked SRAM Object Files (`.pmsf`) and the SRAM Object Files (`.sof`) for each PR region, for each revision. The Assembler creates the `.pmsf` files specifically for partial reconfiguration, one per PR region in each revision.

Table 77. Generated Programming Files

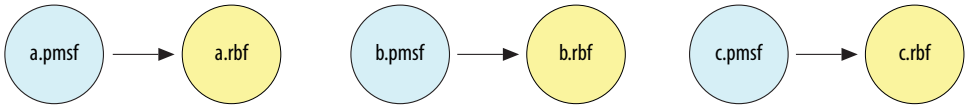
Programming File	Description
<rev>.<pr_region>.pmsf	Contains the partial-mask bits for the PR region. The .pmsf file contains all the information for creating PR bitstreams. *Note:* The default file name corresponds to the partition name.
<rev>.<static_region>.msf	Contains the mask bits for the static region.
<rev>.sof	Contains configuration information for the entire device.

8.2.9.1 Generate PR Bitstreams

After creating the .pmsf files, process the PR bitstreams to generate the Raw Binary File (.rbf) files for reconfiguration. Convert the .pmsf file for every PR region in your design to .rbf file format.

Note: Using the .rbf format stores the bitstream in an external flash memory.

Figure 111. Generating PR Bitstreams



To generate the .rbf file:

1. Click **File ➤ Convert Programming Files**. The **Convert Programming Files** dialog box appears.
2. Specify the **Programming file type** as **Raw Binary File for Partial Reconfiguration (.rbf)**.
3. Specify the output file name.
4. To add the input .pmsf file to convert, click **Add File**.
5. Select the newly added .pmsf file, and click **Properties**.
6. Enable or disable the following options:

- **Compression**—enables compression on PR bitstream.
- **Enhanced compression**—enables enhanced compression on PR bitstream.
- **Generate encrypted bitstream**—generates encrypted independent bitstreams for base image and PR image. You can encrypt the PR image even if your base image has no encryption. The PR image can have a separate encryption key file (`.ekp`).

  Note: Enabling the **Generate encrypted bitstream** option automatically disables **Compression** and **Enhanced compression**. Conversely, enabling **Compression** or **Enhanced compression** automatically disables **Generate encrypted bitstream**. You cannot use compression and encryption at the same time.

  If you enable the **Generate encrypted bitstream** option, specify the following options:

  — **Enable volatile security key**

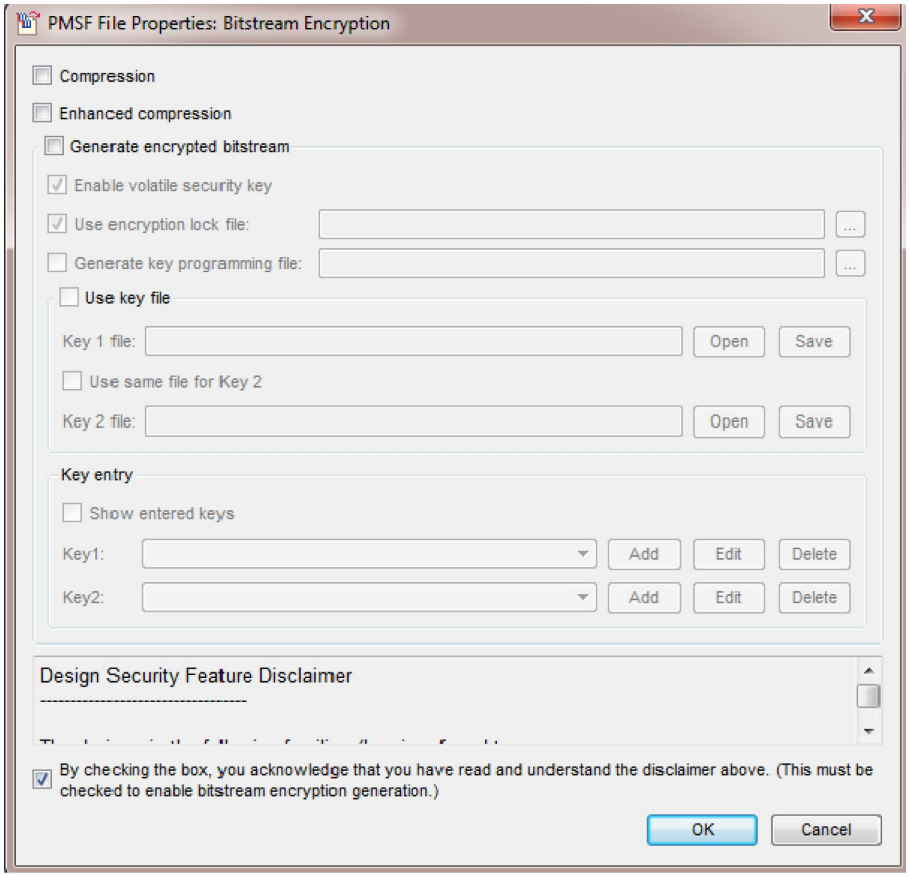
  — **Use encryption lock file**

  — **Generate key programming file**

Note:
- Enabling the **Use encryption lock file** option requires that you import the encryption lock (`.qlk`) file generated from the base image.
- If you configure the device using JTAG, the Programmer does not support base encryption.

7. Click **Generate**.

Figure 112. PMSF File Properties Bitstream Encryption



Alternatively, to convert your `.pmsf` file to `.rbf` file from Quartus Prime shell, type the following command:

```
quartus_cpf -c <pr_pmsf_file> <pr_rbf_file>
```

8.2.9.2 Generating a Merged .pmsf File from Multiple .pmsf Files

Use a single merged `.rbf` file to reconfigure two PR regions simultaneously. To merge two or more `.pmsf` files:

1. Open the **Convert Programming Files** dialog box.
2. Specify the programming file type as **Merged Partial-Mask SRAM Object File (.pmsf)**.
3. Specify the output file name.
4. In the **Input files to convert** dialog box, select **PMSF Data**.
5. To add input files, click **Add File**. You must specify two or more files for merging.
6. To generate the merged file, click **Generate**.

Alternatively, to merge two or more `.pmsf` files from the Quartus Prime shell, type the following command:

```
quartus_cpf --merge_pmsf=<number of merged files>
<pmsf_input_file_1> <pmsf_input_file_2> <pmsf_input_file_etc>
<pmsf_output_file>
```

For example, to merge two `.pmsf` files, type the following command:

```
quartus_cpf --merge_pmsf=<2> <pmsf_input_file_1>
<pmsf_input_file_2> <pmsf_output_file>
```

8.2.9.3 CD Ratio for Bitstream Encryption and Compression

When instantiating the Partial Reconfiguration Controller IP core in your Arria 10 design, you cannot use both data compression and encryption simultaneously. Enhanced decompression uses the same Clock-to-Data (CD) ratio as plain bitstreams (that is, with both encryption and compression off).

The following table lists the valid combinations of bitstream encryption and compression. when enhance compression is enabled, always refer to x16 data width. If you use compression and enhanced compression together, the CD ratio follows the compression bitstream - 4. If you use plain and enhanced compression together, the CD ratio follows the plain bitstream - 1.

Table 78. Valid combinations and CD Ratio for Bitstream Encryption and Compression

Configuration Data Width	AES Encryption	Basic Compression	CD Ratio
x8	Off	Off	1
	Off	On	2
	On	Off	1
x16	Off	Off	1
	Off	On	4
	On	Off	2
x32	Off	Off	1
	Off	On	8
	On	Off	4

The CD ratio for the Partial Reconfiguration IP core must be exact for the bitstream type. The CD ratio for plain RBF must be 1. The CD ratio for compressed RBF must be 2, 4 or 8, depending on the width. Do not specify the CD ratio as the necessary minimum to support different bitstream types.

8.2.9.3.1 Generating an Encrypted PR Bitstream

To partially reconfigure your device with encrypted bitstream:

1. Create a 256-bit key file (`.key`).
2. To generate the key programming file (`.ekp`) from the Quartus Prime shell, type the following command:

```
quartus_cpf --key <keyfile>:<keyid> <base_sof_file> <output_ekp_file>
```

For example:

```
quartus_cpf --key my_key.key:key1 base.sof key.ekp
```

3. To generate the encrypted PR bitstream (`.rbf`), run the following command:

```
quartus_cpf -c <pr_pmsf_file> <pr_rbf_file>
qcrypt -e --keyfile=<keyfile> --keyname=<keyid> –lockto=<qlk file> --
keystore=<battery|OTP> <pr_rbf_file> <pr_encrypted_rbf_file>
```

- `lockto`—specifies the encryption lock.
- `keystore`—specifies the volatile key (battery) or the non-volatile key (OTP).

For example:

```
quartus_cpf -c top_v1.pr_region.pmsf top_v1.pr_region.rbf
qcrypt -e --keyfile=my_key.key --keyname=key1 --keystore=battery
top_v1.pr_region.rbf top_v1_encrypted.rbf
```

4. To program the key file as volatile key (default) into the device, type the following command:

```
quartus_pgm -m jtag -o P;<output_ekp_file>
```

For example:

```
quartus_pgm -m jtag -o P;key.ekp
```

5. To program the base image into the device, type the following command:

```
quartus_pgm -m jtag -o P;<base_sof_file>
```

For example:

```
quartus_pgm -m jtag -o P;base.sof
```

6. To partially reconfigure the device with the encrypted bitstream, type the following command:

```
quartus_pgm -m jtag --pr <output_encrypted_rbf_file>
```

For example:

```
quartus_pgm -m jtag --pr top_v1_encrypted.rbf
```

For more information on the design security features in Arria 10 devices, refer to *Using the Design Security Features in Altera FPGAs*.

Related Links

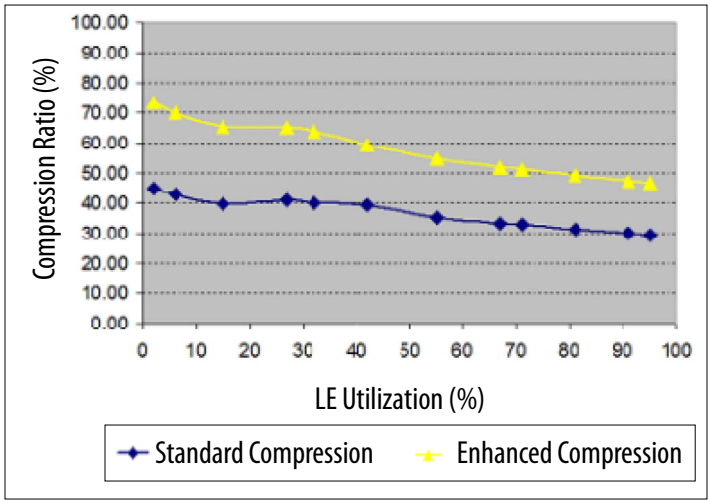
Using the Design Security Features in Altera FPGAs

8.2.9.3.2 Data Compression Comparison

Standard compression results in a 30-45% decrease in RBF size. Use of the enhanced data compression algorithm results in 55-75% decrease in RBF size. The algorithm increases the compression at the expense of additional core area required to implement the compression algorithm.

The following figure shows the compression ratio comparison across PR designs with varying degrees of Logic Element (LE):

Figure 113. Compression Ratio Comparison between Standard Compression and Enhanced Compression



8.2.9.4 Raw Binary Programming File Format

The configuration data in the raw binary programming file (`.rbf`) is little-endian. The following examples show transmitting the `.rbf` byte sequence 02 1B EE 01, in x8, x16, and x32 modes respectively:

Example 75. Writing to the PR control block in x8 mode

In x8 mode, the LSB of a byte is BIT0 and MSB is BIT7.

BYTE0 = 02	BYTE1 = 1B	BYTE2 = EE	BYTE3 = 01
D[7..0]	D[7..0]	D[7..0]	D[7..0]
0000 0010	0001 1011	1110 1110	0000 0001

Example 76. Writing to the PR control block in x16 mode

In x16 mode, the first byte in the file is the least significant byte of the configuration word, and the second byte is the most significant byte of the configuration word.

WORD0 = 1B02		WORD1 = 01EE	
LSB: BYTE0 = 02	MSB: BYTE1 = 1B	LSB: BYTE2 = EE	MSB: BYTE3 = 01
D[7..0]	D[15..8]	D[7..0]	D[15..8]
0000 0010	0001 1011	1110 1110	0000 0001

Example 77. Writing to the PR control block in x32 mode

In x32 mode, the first byte in the file is the least significant byte of the configuration double word, and the fourth byte is the most significant byte.

Double Word = 01EE1B02			
LSB: BYTE0 = 02	BYTE1 = 1B	BYTE2 = EE	MSB: BYTE3 = 01
D[7..0]	D[15..8]	D[23..16]	D[31..24]
0000 0010	0001 1011	1110 1110	0000 0001

8.2.10 Debugging a Partial Reconfiguration Design with System Level Design Tools

Use the Quartus Prime software on-chip debugging tools, such as Signal Tap Logic Analyzer, In-System Sources and Probes Editor (IISP), In-System Memory Content Editor (ISMCE), or JTAG Avalon Master Bridge (JAMB) to verify your partial reconfiguration design.

Note: Only Signal Tap Logic Analyzer allows debugging of both the static and PR regions. Other tools support debugging only in the static region.

Related Links

System Debugging Tools Overview
In *Quartus Prime Pro Edition Handbook Volume 3*

8.2.10.1 Debugging Using Signal Tap Logic Analyzer

Unlike other debugging tools, Signal Tap Logic Analyzer uses the hierarchical debug capabilities provided by Quartus Prime software. This feature allows you to tap signals in the static and PR regions simultaneously. You can debug multiple personas present in your PR region, as well as multiple PR regions. For complete information on the debug infrastructure using hierarchical hubs, refer to *Debugging Partial Reconfiguration Designs Using Signal Tap Logic Analyzer* section in Volume 3 of the Quartus Prime Pro Edition handbook.

Note: The current version of the Quartus Prime Pro Edition software does not support the debug of hierarchical PR regions using Signal Tap Logic Analyzer.

Related Links

Debugging Partial Reconfiguration Designs Using Signal Tap Logic Analyzer

8.2.11 Partial Reconfiguration Simulation and Verification

Simulation verifies the behavior of your design before device programming. The Quartus Prime Pro Edition software supports simulating the delivery of a partial reconfiguration bitstream to the PR control block. This simulation allows you to observe the resulting change and the intermediate effect in a reconfigurable partition.

Similar to non-PR design simulations, preparing for a PR simulation involves setting up your simulator working environment, compiling simulation model libraries, and running your simulation. The Quartus Prime software provides simulation components to help simulate a PR design, and can generate the gate-level PR simulation models for each persona. Use either the behavioral RTL or the gate level PR simulation model for simulation of the PR personas. The gate-level PR simulation model allows for accurate simulation of registers in your design, and reset sequence verification. This technology mapped registers do not assume initial conditions.

Related Links

Simulating Intel FPGA Designs

8.2.11.1 Partial Reconfiguration Simulation Flow

At a high-level, a PR operation consists of the following steps:

1. System-level preparation for a PR event
2. Retrieval of the partial bitstream from memory
3. Transmission of the partial bitstream to the PR control block
4. Resulting change in the design as a new persona becomes active
5. Post-PR system coordination
6. Use of the new persona in the system

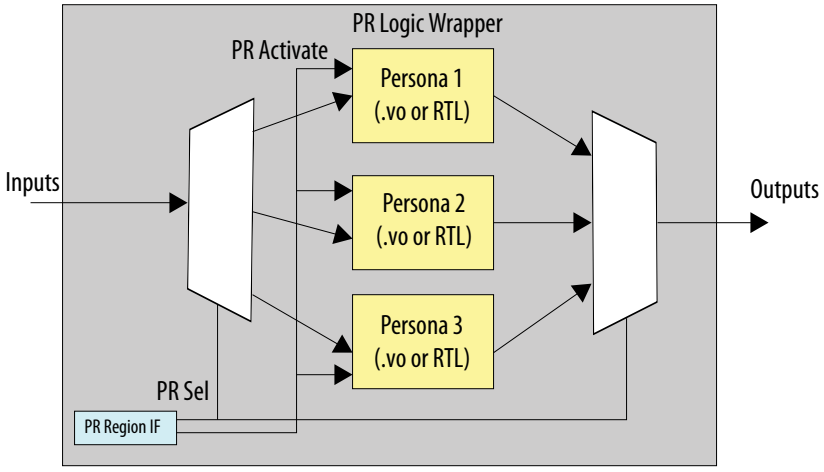
You can simulate each of these process steps in isolation, or as a larger sequence depending on your verification type requirement.

8.2.11.1.1 Simulating PR Persona Replacement

The logical operation of the PR partition changes when a new persona is loaded during the partial reconfiguration process. Simulate the replacement of personas using multiplexors on the input and output of the persona under simulation. Create RTL wrapper logic to represent the top-level of the persona. The wrapper instantiates the default persona during compilation. During simulation, the wrapper allows the replacement of the active persona with another persona. Instantiate each persona as either the behavioral RTL in the PR simulation model the Quartus Prime EDA Netlist Writer generates. The Quartus Prime software includes simulation modules to interface with your simulation testbench:

- `altera_pr_wrapper_mux_in`
- `altera_pr_wrapper_mux_out`
- `altera_pr_persona_if` (SystemVerilog interface allows you to connect the wrapper multiplexes to a testbench driver)

Figure 114. Simulation of PR Persona Switching



Example 78. RTL Wrapper for PR Persona Switching Simulation

The `pr_activate` input of the `altera_pr_wrapper_mux_out` module enables the MUX to output X. This functionality allows the simulation of unknown outputs from the PR persona, and also verifies the normal operation of design's freeze logic. The following code corresponds the simulation of PR persona switching, shown in the above figure:

```
module pr_core_wrapper
(
   input wire a,
   input wire b,
   output wire o
);

localparam ENABLE_PERSONA_1 = 1;
localparam ENABLE_PERSONA_2 = 1;
localparam ENABLE_PERSONA_3 = 1;
localparam NUM_PERSONA = 3;

logic pr_activate;
int persona_select;

altera_pr_persona_if persona_bfm();
assign pr_activate = persona_bfm.pr_activate;
assign persona_select = persona_bfm.persona_select;


wire a_mux [NUM_PERSONA-1:0];
wire b_mux [NUM_PERSONA-1:0];
wire o_mux [NUM_PERSONA-1:0];

generate
   if (ENABLE_PERSONA_1) begin
      localparam persona_id = 0;

      `ifdef ALTERA_ENABLE_PR_MODEL
         assign u_persona_0.altera_sim_pr_activate = pr_activate;
      `endif

      pr_and u_persona_0
      (
         .a(a_mux[persona_id]),
         .b(b_mux[persona_id]),
```

```
         .o(o_mux[persona_id])
      );
   end
endgenerate

generate
   if (ENABLE_PERSONA_2) begin
      localparam persona_id = 1;

      `ifdef ALTERA_ENABLE_PR_MODEL
         assign u_persona_1.altera_sim_pr_activate = pr_activate;
      `endif

      pr_or u_persona_1
      (
         .a(a_mux[persona_id]),
         .b(b_mux[persona_id]),
         .o(o_mux[persona_id])
      );

   end
endgenerate

generate
   if (ENABLE_PERSONA_3) begin
      localparam persona_id = 2;

      `ifdef ALTERA_ENABLE_PR_MODEL
         assign u_persona_2.altera_sim_pr_activate = pr_activate;
      `endif

      pr_empty u_persona_2
      (
         .a(a_mux[persona_id]),
         .b(b_mux[persona_id]),
         .o(o_mux[persona_id])
      );

   end
endgenerate


altera_pr_wrapper_mux_in #(.NUM_PERSONA(NUM_PERSONA), .WIDTH(1))
u_a_mux(.sel(persona_select), .mux_in(a), .mux_out(a_mux));

altera_pr_wrapper_mux_in #(.NUM_PERSONA(NUM_PERSONA), .WIDTH(1))
u_b_mux(.sel(persona_select), .mux_in(b), .mux_out(b_mux));

altera_pr_wrapper_mux_out #(.NUM_PERSONA(NUM_PERSONA), .WIDTH(1))
u_o_mux(.sel(persona_select), .mux_in(o_mux), .mux_out(o), .pr_activate(pr_activat
e));

endmodule
```

PR Simulation Wrapper Modules

`altera_pr_wrapper_mux_in` Module

The `altera_pr_wrapper_mux_in` module allows you to de-multiplex inputs to a PR partition wrapper for all PR personas.

Instantiate one multiplexor per input port. Specify the active persona using the `sel` port of the multiplexor. Parameterize the component to specify the number of persona output and the width of the multiplexor.

```
module altera_pr_wrapper_mux_in(sel, mux_in, mux_out);
    parameter NUM_PERSONA = 1;
    parameter WIDTH = 1;
```

```
    input int sel;
    input wire [WIDTH-1:0] mux_in;
    output reg [WIDTH-1 : 0] mux_out [NUM_PERSONA-1:0];

    always_comb begin
        for (int i = 0; i < NUM_PERSONA; i++)
            if (i == sel)
                mux_out[i] = mux_in;
            else
                mux_out[i] = 'x;
    end


endmodule : altera_pr_wrapper_mux_in
```

The `altera_pr_wrapper_mux_in` component is defined in the `altera_lnsim.sv` file, located in <*QUARTUS_INSTALL_DIR*>`/eda/sim_lib/altera_lnsim.sv`.

`altera_pr_wrapper_mux_out` Module

The `altera_pr_wrapper_mux_out` module allows you to multiplex the outputs of all PR personas to the outputs of the PR region wrapper.

Instantiate one multiplexor per output port. Specify the active persona using the `sel` port of the multiplexor. The optional `pr_activate` port allows you to drive the multiplexor output to "x", to emulate the unknown value of PR region outputs during a PR operation. Parameterize the component to specify the number of persona input and the width of the multiplexor.

```
module altera_pr_wrapper_mux_out(sel, mux_in, mux_out, pr_activate);
    parameter NUM_PERSONA = 1;
    parameter WIDTH = 1;

    input int sel;
    input wire [WIDTH-1 : 0] mux_in [NUM_PERSONA-1:0];
    output reg [WIDTH-1:0]   mux_out;
    input wire               pr_activate;

    always_comb begin
        if ((sel < NUM_PERSONA) && (!pr_activate))
            mux_out = mux_in[sel];
        else
            mux_out = 'x;
    end

endmodule : altera_pr_wrapper_mux_out
```

The `altera_pr_wrapper_mux_out` component is defined in the `altera_lnsim.sv` file, located in <*QUARTUS_INSTALL_DIR*>`/eda/sim_lib/altera_lnsim.sv`.

`altera_pr_persona_if` Module

Instantiate the `altera_pr_persona_if` SystemVerilog interface in a PR region simulation wrapper to connect to all the wrapper multiplexors. Optionally, connect `pr_activate` to the PR simulation model.

Connect the interface's `persona_select` to the `sel` port of all input and output multiplexes. Connect the `pr_activate` to the `pr_activate` of all the output multiplexes. Then, the PR region driver testbench component can drive the interface.

```
interface altera_pr_persona_if;
    logic pr_activate;
    int   persona_select;

    initial begin
```

```
        pr_activate <= 1'b0;
    end
endinterface : altera_pr_persona_if
```

The `altera_pr_persona_if` component is defined in the `altera_lnsim.sv` file, located in *<QUARTUS_INSTALL_DIR>*`/eda/sim_lib/altera_lnsim.sv`.

8.2.11.2 Generating the PR Persona Simulation Model

Use the Quartus Prime EDA Netlist Writer to create the simulation model for a PR persona. The simulation model represents the post-synthesis gate-level netlist for the persona.

When using the PR simulation model for the persona, the netlist includes a new `altera_sim_pr_activate` top-level signal for the model. You can asynchronously drive this signal to load all registers in the model with X. This feature allows you to verify the reset sequence of the new persona on PR event completion. Verify the reset sequence through inspection, using SystemVerilog assertions, or using other checkers.

By default, the PR simulation model asynchronously loads X into the register's storage element on `pr_activate` signal assertion. You can parameterize this behavior on a per register basis, or on a simulation-wide default basis. The simulation model supports four built-in modes:

- `load X`
- `load 1`
- `load 0`
- `load rand`

Specify these modes using the SystemVerilog classes:

- `dffeas_pr_load_x`
- `dffeas_load_1`
- `dffeas_load_0`
- `dffeas_load_rand`

Optionally, you can create your own PR activation class, where your class must define the `pr_load` variable to specify the PR activation value.

Follow these steps to generate the simulation model for a PR design:

1. To run synthesis and generate the simulation netlist for your EDA simulator, click **Synthesis** on the Compilation Dashboard.

Note: The current version of the Quartus Prime software supports the PR simulation model only in SystemVerilog.

2. To generate the PR simulation model, type the following from the command-line:

```
quartus_eda --pr --simulation --tool={your_tool} project -c pr_syn_revision
```

3. To specify a simulation-wide behavior, set the ALTERA_DEFAULT_DFFEAS_PR_ACTIVATE_CLASS macro to the name of the class to use for initialization. For example:

```
define ALTERA_DEFAULT_DFFEAS_PR_ACTIVATE_CLASS dffeas_pr_load_1
```

4. To specify the behavior for an individual register, set the PR_ACTIVATE_CLASS parameter of the specific `dffeas_pr` register to the desired initialization class. For more information, refer to the `dffeas_pr` model in the `altera_lnsim.sv` file, located in *<QUARTUS_INSTALL_DIR>*`/eda/sim_lib/altera_lnsim.sv`.

   Note: The Aldec Riviera-PRO* Simulator does not support selecting different PR_ACTIVATE_CLASS parameters, and only supports registers going to X during `pr_activate`.

Example 79. Built-in Initialization Classes

```
class dffeas_pr_load_x;
    reg pr_load = 1'bx;

    function new();
    endfunction

endclass

class dffeas_pr_load_0;
    reg pr_load = 1'b0;

    function new();
    endfunction

endclass

class dffeas_pr_load_1;
    reg pr_load = 1'b1;

    function new();
    endfunction

endclass

class dffeas_pr_load_rand;
    rand bit pr_load;

    function new(int seed = $random());
        this.srandom(seed);
    endfunction

endclass
```

8.3 Partial Reconfiguration Design Recommendations

When designing for partial reconfiguration, consider the system-level behavior to maintain the integrity and correctness of the static region operation. For example, during PR programming, ensure that the system does not read or write to the PR region. In addition, freeze the write enable output from the PR region into the static region. This freezing avoids interference with the static region operation.

This table lists the partial reconfiguration design guidelines:

Table 79. Partial Reconfiguration Design Guidelines

Scenario	Guideline	Reasoning
Designing for partial reconfiguration	Do not assume initial states in registers. Ensure that you reset all the registers.	The registers contain undefined values after reconfiguration.
	Reset the registers that drive control signals to a known state, after partial reconfiguration.	Registers contain undefined values after reconfiguration. In addition, synthesis can duplicate registers.
	You cannot define synchronous reset as a global signal for partial reconfiguration.	PR regions do not support synchronous reset of registers as a global signal, because the Arria 10 LAB does not support synchronous clear (`sclr`) signal on a global buffer. The LAB supports the asynchronous clear (`aclr`) signal driven from a local input, or from a global network row clock. As a result, only the `aclr` can be a global signal, feeding registers in a PR region.
Partitioning the design	Register all the inputs and outputs for your PR region.	Improves timing closure and time budgeting.
	Reduce the number of signals interfacing the PR region with the static region in your design.	Reduces the wire LUT count.
	Create a wrapper for your PR region.	The wrapper creates common footprint to static region.
	Drive all the PR region output ports to inactive state.	Prevents the static region logic from receiving random data during the partial reconfiguration operation.
	PR boundary I/O interface must be a superset of all the PR persona I/O interfaces.	Ensures that each PR partition implements the same ports.
Preparing for partial reconfiguration	Complete all pending transactions.	Ensures that the static region is not in a wait state.
Maintaining a partially working system during partial reconfiguration	Hold all outputs to known constant values.	Ensures that the undefined values received from the PR region during and after the reconfiguration does not affect the PR control logic.
Initializing after partial reconfiguration	Initialize after reset.	Retrieves state from memory or other device resources.

continued...

Scenario	Guideline	Reasoning
Debugging the design using Signal Tap Logic Analyzer	• Do not tap signals in the default personas. • Store all the tapped signals from a persona in one `.stp` file.	The current version of the Quartus Prime software supports only one `.stp` (signal tap file) per revision. This limitation requires you to select partitions, one at a time, to tap.
	Do not tap across regions in the same `.stp` file.	Ensures consistent interface (boundary) across all personas.
	Tap only the pre-synthesis signals. In the Node Finder, filter for **Signal Tap: pre-synthesis**.	Ensures that the signal tapping of PR personas start from synthesis.

8.4 Partial Reconfiguration Design Considerations

Partial reconfiguration is an advanced design flow within the Quartus Prime Pro Edition software. Successfully creating a partial reconfiguration design requires understanding the requirements and design practices of the PR flow.

The following list summarizes the design considerations for partial reconfiguration:

- Reconfigurable partitions can only contain core resources, such as LABs, RAMs, and DSPs. All periphery resources, such as the transceivers, external memory interface, HPS, and clocks must be in the static portion of the design.
- To physically partition the device between static and individual PR regions, floorplan each PR region into exclusive, core-only, placement regions, with associated routing regions.
- A reconfiguration partition must contain the super-set of all ports that you use across all PR personas.
- To minimize programming files size, ensure that the PR regions are short and wide.
- The maximum number of clocks or other global signals for any PR region is 33. In the current version of the Quartus Prime Pro Edition software, no two PR regions can share a row-clock.
- In Arria 10 devices, the PR regions do not require any input freeze logic. However, you must freeze all the outputs of each PR region to a known constant value to avoid unknown data during partial reconfiguration.
- Your PR design must consider all the system-level coordination of partial reconfiguration.
- The current version of the Quartus Prime Pro Edition software supports only one `.stp` (signal tap file) per revision. For designs with multiple PR regions, generate one revision for each PR region you wish to debug.
- Increase the reset length by 1 cycle to account for register duplication in the Fitter.
- All Arria 10 devices in -1, -2 and -3 speed grade support partial reconfiguration.
- Use the nominal VCC of 0.9V or 0.95V as per the datasheet, including VID enabled devices.
- Quartus Prime Standard Edition software does not support partial reconfiguration for Arria 10 devices.

8.5 Document Revision History

This document has the following revision history.

Table 80. Document Revision History

Date	Version	Changes
2017.05.08	17.0.0	• Added information about Hierarchical Partial Reconfiguration. • Added new topic 'Partial Recinfiguration Simulation and Verification'. • Added new topic 'Run Timing Analysis on a Design with Multiple PR Partitions'. • Updated topic 'Freeze Logic for PR Regions'. • Added new topic 'Debugging Using Signal Tap Logic Analyzer'. • Other minor updates.
10.31.2016	16.1.0	• Implemented Intel rebranding. • Initial release.

Related Links

Altera Documentation Archive

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

9 Creating a System With Qsys Pro

Qsys Pro is a system integration tool included as part of the Quartus Prime software. Qsys Pro simplifies the task of defining and integrating customized IP Components (IP Cores) into your designs.

Qsys Pro facilitates design reuse by packaging and integrating your custom IP components with Intel and third-party IP components. Qsys Pro automatically creates interconnect logic from the high-level connectivity that you specify, which eliminates the error-prone and time-consuming task of writing HDL to specify system-level connections.

Qsys Pro introduces hierarchical isolation between system interconnect and IP components. Qsys Pro stores the instantiated IP component in a separate `.ip` file and the system connectivity information in the `.qsys` file. This hierarchical isolation ensures that changing the parameters of a single IP component does not necessitate regeneration of the enclosing system or any other IP component within that system. Likewise, a change to system connectivity does not require regeneration of any of the IP components. Qsys Pro references the parameterized IP component for instantiation in the system by the component's entity name, and generates the RTL of the IP component and the RTL of the system separately.

Qsys Pro is a more powerful tool if you design your custom IP components using standard interfaces available in the Qsys Pro IP Catalog. Standard interfaces inter-operate efficiently with the Intel FPGA IP components, and you can take advantage of bus functional models (BFMs), monitors, and other verification IP to verify your systems.

Qsys Pro supports Avalon®, AMBA® AXI3™ (version 1.0), AMBA AXI4™ (version 2.0), AMBA AXI4-Lite™ (version 2.0), AMBA AXI4-Stream (version 1.0), and AMBA APB™3 (version 1.0) interface specifications.

Qsys Pro provides the following advantages:

- Simplifies the process of customizing and integrating IP components into systems
- Provides isolation between the system and IP component, maintaining all the parameter information of the IP component in a separate `.ip` file.
- Supports generic components, allowing the instantiation of IP components without an HDL implementation.
- Generates an IP core variation for use in your Quartus Prime software projects
- Supports incremental generation of the system and IP components.
- Allows specifying interface requirements for the system.
- Supports up to 64-bit addressing
- Supports modular system design
- Supports visualization of systems

- Supports optimization of interconnect and pipelining within the system
- Supports auto-adaptation of different data widths and burst characteristics
- Supports inter-operation between standard protocols, such as Avalon and AXI
- Fully integrated with the Quartus Prime software

Note: For information on how to define and generate stand-alone IP cores for use in your Quartus Prime software projects, refer to *Introduction to Intel FPGA IP Cores* and *Managing Quartus Prime Projects*.

Related Links

- Introduction to Intel FPGA IP Cores
- Managing Quartus Prime Projects on page 31
  The Quartus Prime software organizes and manages the elements of your design within a *project*.
- Avalon Interface Specifications
- AMBA Protocol Specifications

9.1 Interface Support in Qsys Pro

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

Qsys Pro IP components can include the following interface types:

Table 81. IP Component Interface Types

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

continued...

Interface Type	Description
	level of the system as top-level system I/O. You can use conduits to connect to external devices, for example external DDR SDRAM memory, and to FPGA logic defined outside of the Qsys Pro system.

9.2 Introduction to the Qsys Pro IP Catalog

The Qsys Pro IP Catalog offers a broad range of configurable IP Cores optimized for Intel devices to use in your Qsys Pro designs.

The Quartus Prime software installation includes the Intel FPGA IP library. You can integrate optimized and verified Intel FPGA IP cores into your design to shorten design cycles and maximize performance. The IP Catalog can include Intel-provided IP components, third-party IP components, custom IP components that you create in the Qsys Pro Component Editor, and previously generated Qsys Pro systems.

The Qsys Pro IP Catalog includes the following IP component types:

- Microprocessors, such as the Nios II processor
- DSP IP cores, such as the Reed Solomon Decoder II
- Interface protocols, such as the IP Compiler for PCI Express
- Memory controllers, such as the RLDRAM II Controller with UniPHY
- Avalon Streaming (Avalon-ST) IP cores, such as the Avalon-ST Multiplexer
- Qsys Pro Interconnect
- Verification IP (VIP) Bus Functional Models (BFMs)

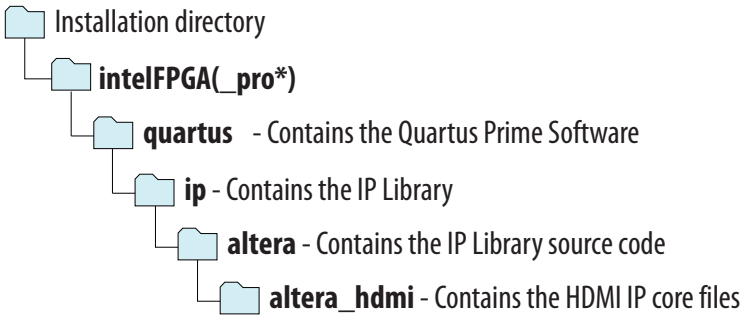
Related Links

Introduction to Intel FPGA IP Cores

9.2.1 Installing and Licensing IP Cores

The Quartus Prime software includes the Intel FPGA IP Library. The library provides many useful IP core functions for production use without additional license. You can fully evaluate any licensed Intel FPGA IP core in simulation and in hardware until you are satisfied with its functionality and performance. The HDMI IP core is part of the Intel FPGA IP Library, which is distributed with the Quartus Prime software and downloadable from www.altera.com.

Figure 115. HDMI Installation Path



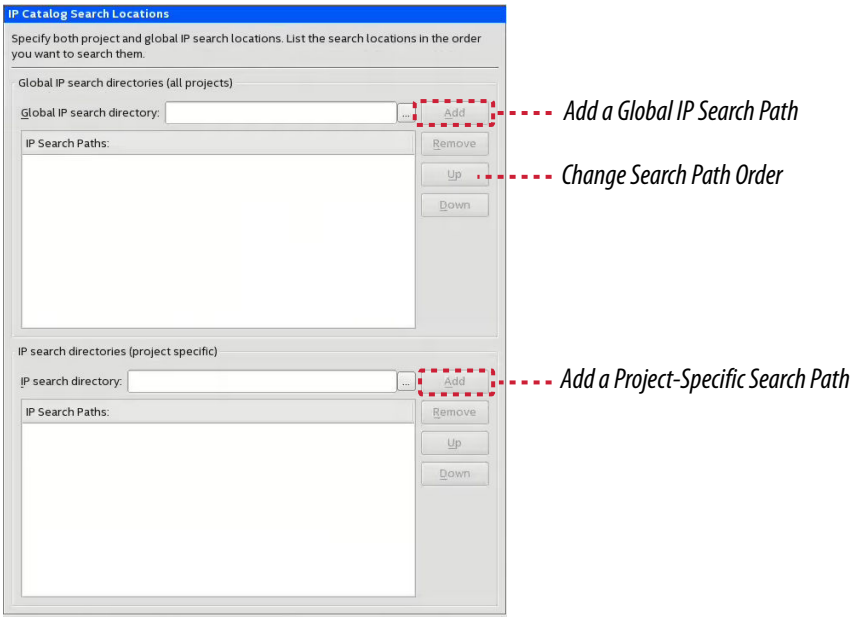
Note: The default IP installation directory on Windows* is `<drive>:\intelFPGA_pro\quartus\ip\altera`; on Linux* it is `<home directory>/intelFPGA_pro/quartus/ip/altera`.

After you purchase a license for the HDMI IP core, you can request a license file from the licensing site and install it on your computer. When you request a license file, Intel emails you a `license.dat file`. If you do not have Internet access, contact your local Intel representative.

9.2.2 Adding IP Cores to IP Catalog

The IP Catalog automatically displays IP cores located in the project directory, in the default Quartus Prime installation directory, and in the IP search path.

Figure 116. Specifying IP Search Locations



The IP Catalog displays Quartus Prime IP components and Qsys Pro systems, third-party IP components, and any custom IP components that you include in the path. Use the **IP Search Path** option (**Tools ➤ Options**) to include custom and third-party IP components in the IP Catalog.

The Quartus Prime software searches the directories listed in the IP search path for the following IP core files:

- Component Description File (`_hw.tcl`)—defines a single IP core.
- IP Index File (`.ipx`)—each `.ipx` file indexes a collection of available IP cores. This file specifies the relative path of directories to search for IP cores. In general, `.ipx` files facilitate faster searches.

The Quartus Prime software searches some directories recursively and other directories only to a specific depth. When the search is recursive, the search stops at any directory that contains a `_hw.tcl` or `.ipx` file.

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

Table 82. IP Search Locations

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

If the Quartus Prime software recognizes two IP cores with the same name, the following search path precedence rules determine the resolution of files:

1. Project directory.
2. Project database directory.
3. Project IP search path specified in **IP Search Locations**, or with the SEARCH_PATH assignment for the current project revision.
4. Global IP search path specified in **IP Search Locations**, or with the SEARCH_PATH assignment in the `quartus2.ini` file.
5. Quartus software libraries directory, such as <*`Quartus Installation`*>`\libraries`.

Note: If you add an IP component to the search path, update the IP Catalog by clicking **Refresh IP Catalog** in the drop-down list. In Qsys and Qsys Pro, click **File ➤ Refresh System** to update the IP Catalog.

9.2.3 General Settings for IP

Use the following settings to control how the Quartus Prime software manages IP cores in your project.

Table 83. IP Core General Setting Locations

Setting Location	Description
Tools ➤ Options ➤ IP Settings Or **Tasks pane ➤ Settings ➤ IP Settings** (Pro Edition Only)	• Specify the **IP generation HDL preference**. The parameter editor generates the HDL you specify for IP variations. • Increase the **Maximum Qsys memory usage size** if you experience slow processing for large systems, or for out of memory errors. • Specify whether to **Automatically add Quartus Prime IP files** to all projects. Disable this option to manually add the IP files. • Use the **IP Regeneration Policy** setting to control when synthesis files regenerate for each IP variation. Typically, you **Always regenerate synthesis files for IP cores** after making changes to an IP variation.
Tools ➤ Options ➤ IP Catalog Search Locations Or **Tasks pane ➤ Settings ➤ IP Catalog Search Locations** (Pro Edition Only)	• Specify additional project and global IP search locations. The Quartus Prime software searches for IP cores in the project directory, in the Quartus Prime installation directory, and in the IP search path.

9.2.4 Set up the IP Index File (.ipx) to Search for IP Components

An IP Index File (**.ipx**) contains a search path that Qsys Pro uses to search for IP components. You can use the `ip-make-ipx` command to create an **.ipx** file for any directory tree, which can reduce the startup time for Qsys Pro.

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

A <*path*> element in the **.ipx** file specifies a directory where multiple IP components may be found. A <*component*> entry specifies the path to a single component. A <*path*> element can use wildcards in its definition. An asterisk matches any file name. If you use an asterisk as a directory name, it matches any number of subdirectories.

Example 80. Path Element in an .ipx File

```
<library>
        <path path="…<user directory>" />
        <path path="…<user directory>" />
        …
        <component … file="…<user directory>" />
        …
</library>
```

A <*component*> element in an **.ipx** file contains several attributes to define a component. If you provide the required details for each component in an **.ipx** file, the startup time for Qsys Pro is less than if Qsys Pro must discover the files in a directory. The example below shows two <*component*> elements. Note that the paths for file names are specified relative to the **.ipx** file.

Example 81. Component Element in an .ipx File

```
<library>
  <component
    name="A Qsys Pro Component"
    displayName="Qsys Pro FIR Filter Component"
    version="2.1"
    file="./components/qsys_filters/fir_hw.tcl"
   />
  <component
    name="rgb2cmyk_component"
    displayName="RGB2CMYK Converter(Color Conversion Category!)"
    version="0.9"
    file="./components/qsys_converters/color/rgb2cmyk_hw.tcl"
   />
</library>
```

Note: You can verify that IP components are available with the `ip-catalog` command.

Related Links

Create an .ipx File with ip-make-ipx on page 374
The `ip-make-ipx` command creates an .**ipx** index file. This file provides a convenient way to include a collection of IP components from an arbitrary directory. You can edit the .**ipx** file to disable visibility of one or more IP components in the IP Catalog.

9.2.5 Integrate Third-Party IP Components into the Qsys Pro IP Catalog

You can use IP components created by Intel partners in your Qsys Pro systems. These IP components have interfaces that are supported by Qsys Pro, such as Avalon-MM or AXI. Additionally, some include timing and placement constraints, software drivers, simulation models, and reference designs.

To locate supported third-party IP components on Altera's web page, navigate to the *Intellectual Property & Reference Designs* page, type `Qsys Pro Certified` in the **Search** box, select **IP Core & Reference Designs**, and then press **Enter**.

Refer to Intel's *Intellectual Property & Reference Designs* page for more information.

Related Links

Intellectual Property & Reference Designs

9.3 Create a Qsys Pro System

Click **Tools ➤ Qsys Pro** in the Quartus Prime software to open Qsys Pro. A `.qsys` file represents your Qsys Pro system in your Quartus Prime software project.

Related Links

- Creating Qsys Pro Components on page 585
  You can create a Hardware Component Definition File (`_hw.tcl`) to describe and package IP components for use in a Qsys Pro system.
- Component Interface Tcl Reference on page 762
  Tcl commands allow you to perform a wide range of functions in Qsys Pro. Command descriptions contain the Qsys Pro phases where you can use the command, for example, main program, elaboration, composition, or fileset callback.

9.3.1 Create/Open Project in Qsys Pro

The Quartus Prime software tightly links with Qsys Pro system creation. Qsys Pro requires you to specify a Quartus Prime project at time of system creation.

To create a new system, or open an existing system in Qsys Pro:

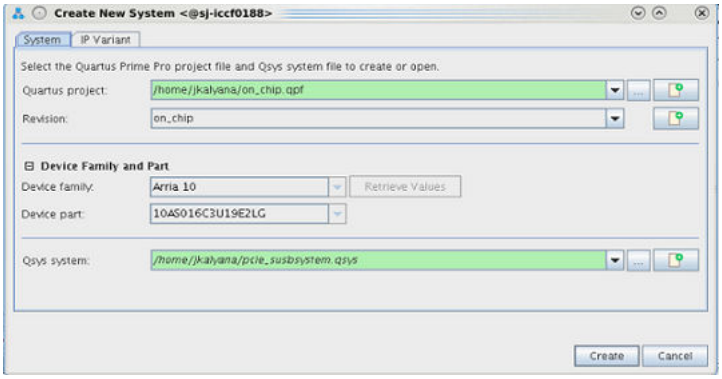
1. To create a new Quartus Prime project to associate with your Qsys Pro system, click  . To select an existing project, browse for the project. Alternatively, select an existing project from the drop-down list in the **Quartus project** field.

Note: Selecting **None** from the drop-down list in the **Quartus project** field opens the Qsys Pro tool in view-only mode.[4]

2. To create a new revision for the Quartus Prime project, click . To specify an existing revision for the project, select an existing revision from the drop-down list in the **Revision** field.
3. When creating a new Quartus Prime project, specify the **Device family** and **Device part** to associate with your Qsys Pro system by selecting the device name and device part number from the respective fields. If you are opening an existing Quartus Prime project to associate with your Qsys Pro system, click **Retrieve Values** to populate the fields with the device information of the Quartus Prime project.
4. To create a new Qsys Pro system, click . To open an existing `.qsys` file, browse for the file. Alternatively, select an existing file from the drop-down list.

Note: Similarly, you can open an existing IP file, or create a new IP variant by selecting the **IP Variant** tab in the **Create New System** dialog box. To create a new IP variant, you must specify a **Component type** for the `.ip` file.

Figure 117. Qsys Pro Create New System



Note:

- To change the Quartus Prime project associated with your current Qsys Pro system, click **File ➤ Select Quartus Project**.

9.3.1.1 Convert your Existing System to Qsys Pro Format

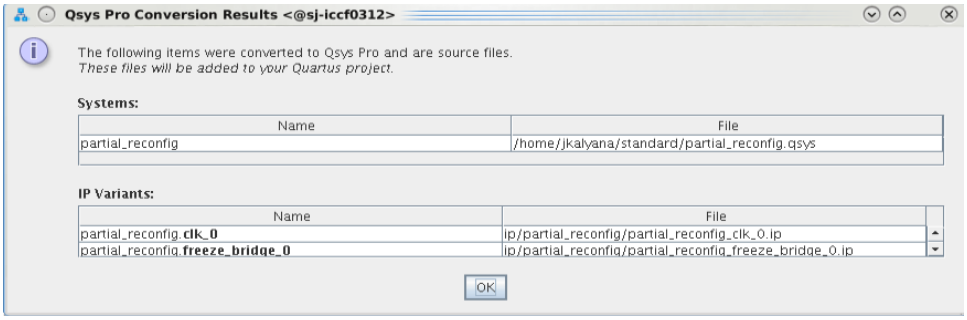
When you open an existing system with incompatible components, Qsys Pro prompts you to convert these components to the Qsys Pro format. On conversion, the **Qsys Pro Conversion Results** dialog box appears, listing all the converted system and IP source files.

---

4 View-only mode restricts the following functionality:

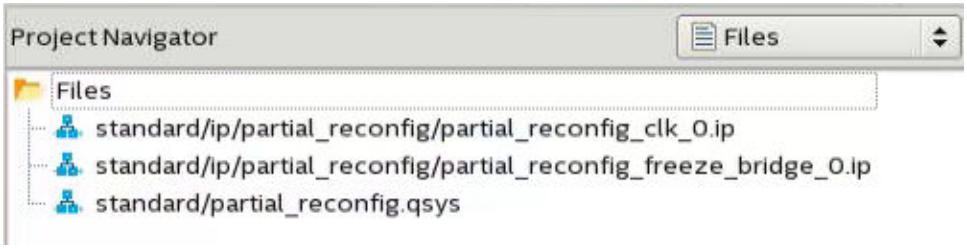
- Adding new IP components to the system or subsystem.
- Removing the instantiated IP components from the system or subsystem.
- Creating a new system, subsystem, or IP file.
- Executing system scripts.

Figure 118. Qsys Pro Conversion Results Dialog Box



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

Figure 119. System and IP Files Associated with a Quartus Prime Project



9.3.2 Modify the Target Device

The Qsys Pro system inherits the device family from the associated Quartus Prime project.

You can modify the device settings of your Qsys Pro system from the **Device Family** tab. Changing the **Device family** or **Device** options from this tab automatically updates the associated Quartus Prime project.

9.3.3 Modify the IP Search Path

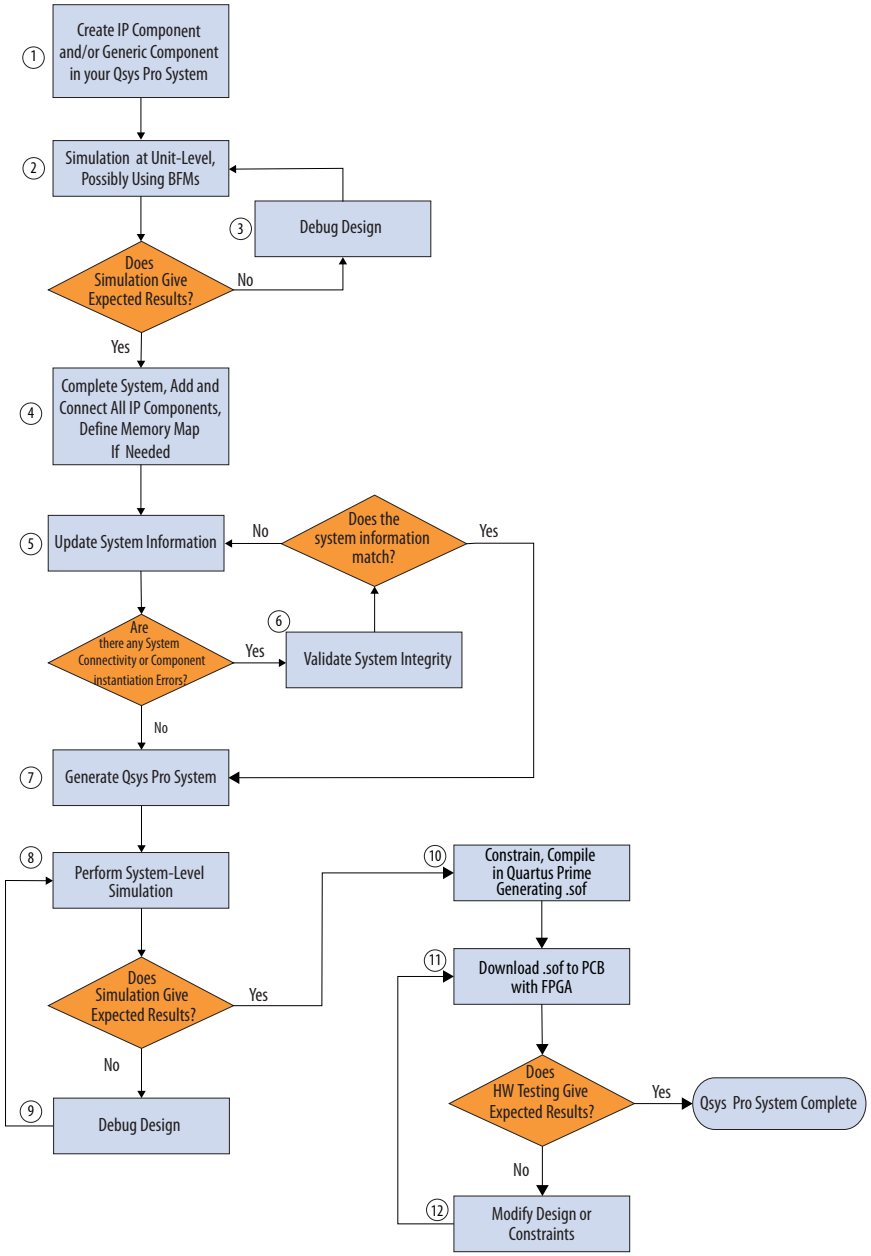
Qsys Pro allows you to view and modify the IP search locations specified for the Quartus Prime project associated with your system. To specify the IP search path from Qsys Pro:

1. Click **Tools ➤ Options ➤ IP Search Path**. The Quartus Prime Global IP Search Path and Quartus Project IP Search Path panes display the IP search locations specified for your associated Quartus Prime project.
2. Click **Add** or **Remove** to add/remove new search locations. The Quartus Prime project automatically updates to reflect these modifications. In Qsys Pro, click **File ➤ Refresh System** to propagate these changes.

9.3.4 Qsys Pro System Design flow

The Qsys Pro design flow involves creating, instantiating and generating, and simulating system output for IP components.

Figure 120. Qsys Pro System Design Flow



Note: For information on how to define and generate single IP cores for use in your Quartus Prime software projects, refer to *Introduction to Intel FPGA IP Cores*.

Related Links

Introduction to Intel FPGA IP Cores

9.3.5 Add IP Components (IP Cores) to a Qsys Pro System

The Qsys Pro IP Catalog displays IP components (IP cores) available for your target device. Double-click any component in the IP Catalog to launch the parameter editor. The parameter editor allows you to create a custom IP component variation of the selected component. A Qsys Pro system can contain a single instance of an IP component, or multiple, individually parameterized variations of multiple or the same IP components.

Qsys Pro preserves each of the IP component's parameters as a `.ip` file. A Qsys Pro system instantiates a generic component in place of the actual IP core with a reference to the HDL entity name, module and interface assignments, compilation library, HDL ports, interfaces, and system-info parameters.

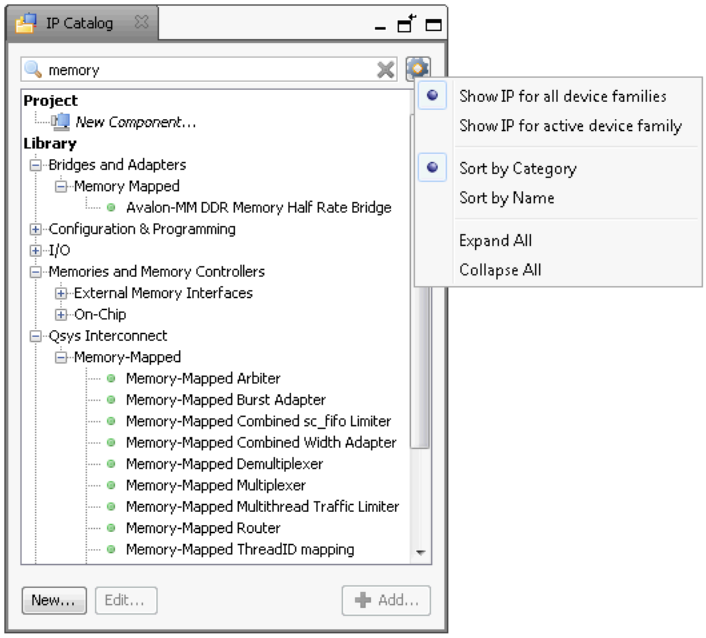
Follow these steps to locate, instantiate, and customize an IP component in your Qsys Pro system:

1. Right-click any IP component name in the Qsys Pro IP Catalog to display details about device support, installation location, versions, and links to documentation.
2. To locate a specific type of component, type some or all of the component's name in the **IP Catalog** search box.
   For example, type **memory** to locate memory-mapped IP components, or **axi** to locate AXI IP. You can also filter the IP Catalog display with options on the right-click menu.
3. To launch the parameter editor, double-click any component. You can set the parameter values in the parameter editor and view the block diagram for the component. The **Parameterization Messages** tab at the bottom displays any errors in the parameterization of the IP component.
4. For IP components that have preset parameter values, select the preset file in the preset editor, and then click **Apply**. This option allows you to instantly apply preset parameter values for the IP component appropriate for a specific application.
5. To complete customization of the IP component, click **Finish**. The IP component appears in the **System Contents** and **Component Instantiation** tabs.

   Note: Qsys Pro creates a corresponding `.ip` file for the IP component on instantiation, and stores the file in the `<ip>` folder in your project directory.

The IP component appears in the **System Contents** tab.

Figure 121. Qsys Pro IP Catalog



9.3.6 Specify Implementation Type for IP Components

A Qsys Pro system instantiates a generic component in place of the actual IP core with a reference to the HDL entity name, module and interface assignments, compilation library, HDL ports, interfaces, and system-info parameters.

The **Component Instantiation** tab allows you to configure the system representation of an IP core. To open the **Component Instantiation** tab, click **View ➤ Component Instantiation.**

Table 84. Component Instantiation GUI Information

Name	Description
Implementation Type	Allows you to decide how to define the implementation of your IP component. Qsys Pro has the following implementation types:

continued...

Name	Description
	• **IP**—The default implementation type for any IP core. With **IP Implementation Type**, Qsys Pro performs the following functions: — Runs background checks against the port widths between the IP component and the `.ip` file to ensure continuity. — Scans the `.ip` file for the error flag to understand if any component has parameterization errors. — Checks for system-info mismatches between the IP file and the IP component in the system and prompts you to resolve these through IP instantiation warnings in the **Instantiation Messages** tab. • **HDL**—Allows you to quickly import RTL to your Qsys Pro system. You can populate the signals and interfaces parameters of the generic component from an RTL file. • **Blackbox**—By choosing this implementation type, you specify a component that represents the signal and interface boundary of an entity, without providing the component's implementation. You must provide the implementation of the component in a downstream compiler such as Quartus Prime software or your RTL simulator.
Compilation Info	Allows you to specify the **HDL Entity name** and **HDL compilation library** name for the implementation. These are fixed values for the **IP Implementation Type**.
Signals & Interfaces	Allows you to define the port boundary of the component. Click ***<<add interface>>*** or ***<<add signal>>*** to add the interfaces and signals.
System Information	Allows you to specify the address map of the interfaces, input clock rate, and other necessary system information associated with the component.
Block symbol	Allows you to visualize the signals and interfaces added in the **Signals & Interfaces** tab.
Implementation Templates	Allows you to export implementation templates in the form of a pre-populated HDL entity, or a template Qsys Pro system which contains the boundary information (signals and interfaces) as interface requirements.
Export	Allows you to export the signals and interfaces of an IP component as an IP-XACT file or a `_hw.tcl` file.

Note: Remember to click **Apply** in the **Component Instantiation** tab for any of your changes to take effect. Alternatively, click **Revert** to undo all the changes you have made to the component.

Related Links

Adding a Generic Component to the Qsys Pro System on page 619
The generic component is a new type of Qsys component that enables hierarchical isolation of IP components.

9.3.7 Connect IP Components in Your Qsys Pro System

Use the **System Contents** tab to connect and configure components. Qsys Pro supports connections between interfaces of compatible types and opposite directions. For example, you can connect a memory-mapped master interface to a slave interface, and an interrupt sender interface to an interrupt receiver interface. You can connect any interfaces exported from a Qsys Pro system within a parent system.

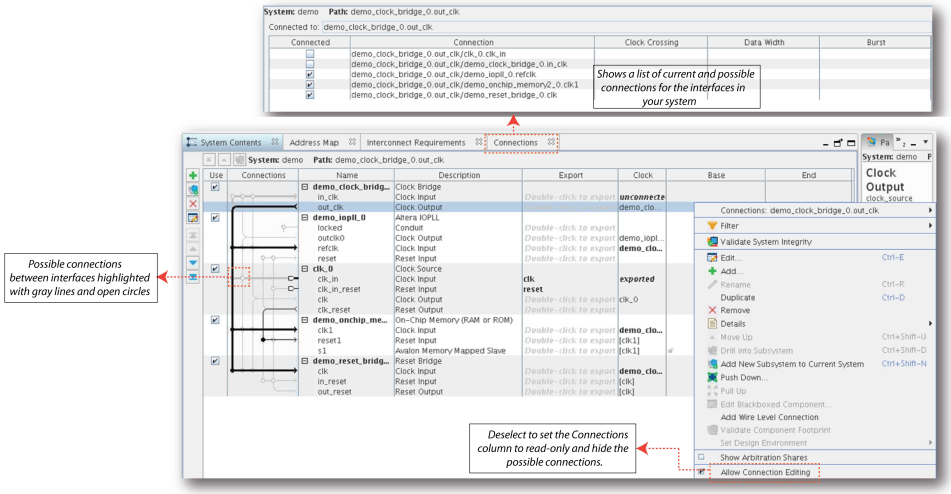
Note: You cannot both export and connect signals internally within the same Qsys Pro system.

Possible connections between interfaces appear as gray lines and open circles. To make a connection, click the open circle at the intersection of the interfaces. When you make a connection, Qsys Pro draws the connection line in black and fills the connection circle. Clicking a filled-in circle removes the connection.

Qsys Pro takes the high-level connectivity you specify, and instantiates a suitable HDL fabric to perform the needed adaptation and arbitration between components. Qsys Pro includes this interconnect fabric in the generated RTL system output. The **Connections** tab (**View ➤ Connections**) shows a list of current and possible connections for selected instances or interfaces in the **Hierarchy** or **System Contents** tabs. You can add and remove connections by clicking the check box for each connection. Each column provides specific information about the connection. For example, the **Clock Crossing**, **Data Width**, and **Burst** columns provide interconnect information about added adapters that can result in slower f_{MAX} or increased area utilization.

Figure 122. Connections Column in the System Contents Tab

To prevent additional connectivity changes to your system, you can deselect **Allow Connection Editing** in the right-click menu. This option sets the **Connections** column to read-only and hides the possible connections.



9.3.7.1 Create Connections Between Masters and Slaves

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

Qsys Pro enables you to design a system where two masters access the same slave at different addresses. If you use this feature, Qsys Pro labels the **Base** and **End** address columns in the **System Contents** tab as "mixed" rather than providing the address range.

Follow these steps to change or create a connection between master and slave IP components:

1. In Qsys Pro, click the **Address Map** tab.
2. Locate the table cell that represents the connection between the master and slave component pair.
3. Either type in a base address, or update the current base address in the cell.

Note: The base address of a slave component must be a multiple of the address span of the component. This restriction is a requirement of the Qsys interconnect, which provides an efficient address decoding logic, which in turn allows Qsys Pro to achieve the best possible f_{MAX}.

9.3.8 Validate System Integrity

The **System Messages** tab displays all the errors and warnings associated with your current Qsys Pro system. Double-click the warning or error messages to open the relevant **System Contents** or **Parameters** tabs to fix the issue. You can also click validate button in the **Hierarchy** tab, or the **Validate System Integrity** button at the bottom of the main Qsys Pro panel to perform system integrity check for the entire system.

Table 85. System Messages Types in Qsys Pro

System Messages Types	Description
Component Instantiation Warning	Indicates the mismatches between system information parameters or IP core parameterization errors. A system information parameters mismatch refers to the mismatch between an IP component's system parameter expectations and the component's saved system information parameters in the corresponding `.ip` file.
Component Instantiation Error	Indicates the mismatches between HDL entity name, compilation library, or ports which results in downstream compilation errors. The component instantiation errors always indicate the fundamental mismatches between generated system and interconnect fabric RTL.
System Connectivity Warning	Qsys system connectivity warnings.
System Connectivity Error	Qsys system connectivity errors.

9.3.8.1 Component Instantiation Warning Messages

Component Instantiation Warnings report the following inconsistencies:

- Interface types do not match
- Interface is missing
- Port has been moved to another interface
- Port role has changed
- Interface assignment is mismatched
- Interface assignment is missing

9.3.8.2 Component Instantiation Error Messages

Component Instantiation Errors report the following inconsistencies:

- Port is missing from the ip file
- Port is missing from instantiation
- Port direction has changed
- Port VHDL type has changed
- Port width has changed
- Interface Parameter is mismatched
- Interface Parameter is missing

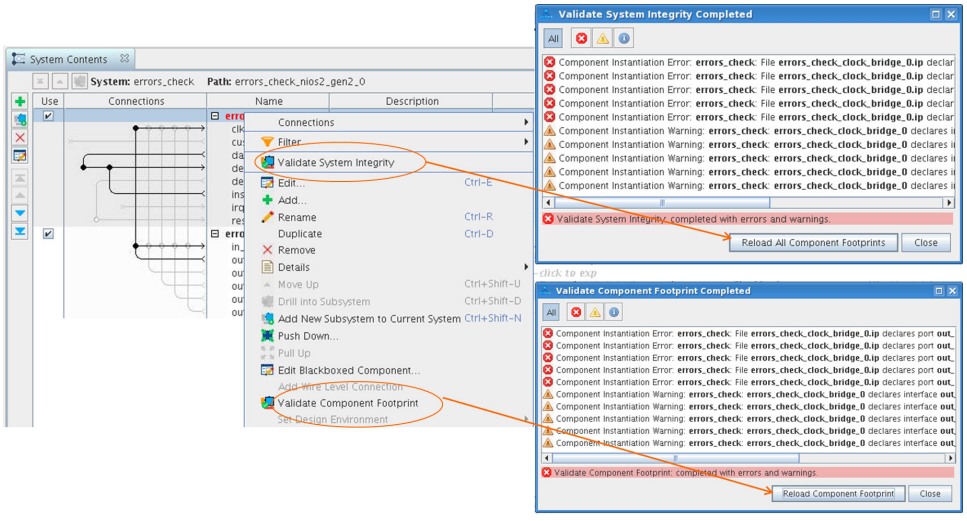
9.3.8.3 Validate System Integrity for Individual Components in the System

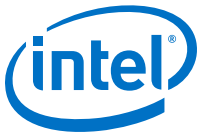
To validate the system integrity for your IP components:

1. Select the IP component in the **System Contents** tab.
2. Right-click and select **Validate Component Footprint** to check for any mismatches between the IP component and its `.ip` file representation.
3. If there are any errors, click **Reload Component Footprint** to reload the signals and interfaces for the component from the `.ip` file.

Note: To perform system integrity check for the entire system, right-click the **System Contents** tab and select **Validate System Integrity**. You can also click the validate button in the **Hierarchy** tab, or the **Validate System Integrity** button at the bottom of the main Qsys Pro panel.

Figure 123. Validating System Integrity





9.3.9 Propagate System Information to IP Components

When system information doesn't match the requirements of an IP component, use the **System Info** tab to synchronize the IP component with mismatches. To open the **System Info** tab, click **View ➤ System Info**.

Table 86. System Info GUI Information

Name	Description
Component Instantiation	This table shows the signals and interfaces for the selected IP component within the system. Mismatches are highlighted in blue. Missing elements are highlighted in green.
IP file	This table shows the signal and interface information for the selected IP component from its corresponding `.ip` file. Mismatches are highlighted in blue. Missing elements are highlighted in green.
Component Instantiation Value	This table shows the selected interface parameter value of the IP component within the system.
IP File Value	This table shows the selected interface parameter value of the IP component from the corresponding `.ip` file.
>>	This button allows you to manually synchronize the mismatches in signals and interfaces, one at a time, between the IP file and the IP component.
Sync All	This button allows you to synchronize all the system info mismatches for the IP component.

Note: To update the system information for all the IP components in your current system simultaneously, click the update icon in the **Hierarchy** tab or the **Sync All System Infos** button at the bottom of the main Qsys Pro panel.

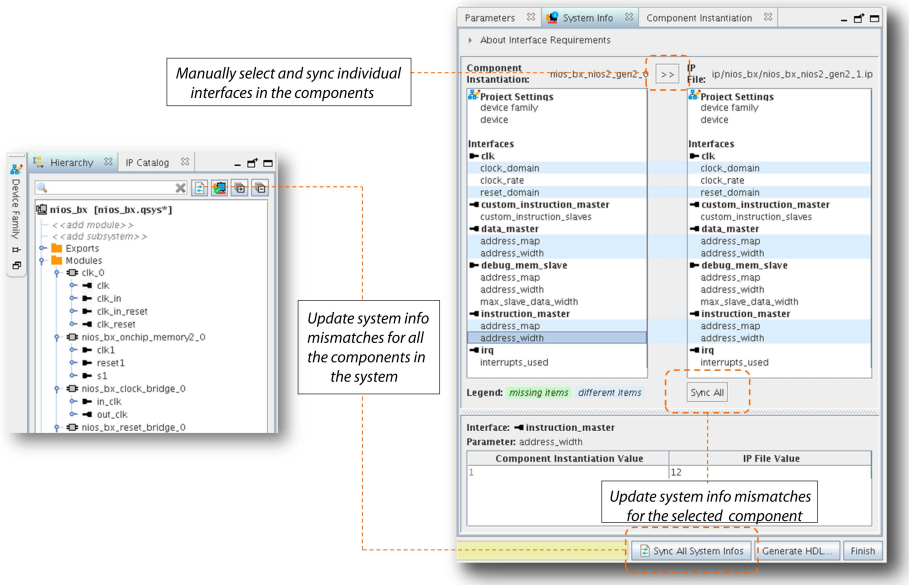
9.3.9.1 Update System Information

If the system information does not match the saved requirements of the corresponding `.ip` file for an IP component, the mismatches appear as Component Instantiation Warnings in the **System Messages** tab. In Qsys Pro, you must manually synchronize these system info dependencies:

1. To open the **System Info** tab, select the signal or interface in the **System Contents** tab and click **View ➤ System Info**. You can also double-click the corresponding Component Instantiation Warning in the **System Messages** tab to open the system-info mismatch information in the **System Info** tab.
2. To update the `.ip` file with the current system information, select the mismatched parameter and click **>>**. Alternatively, you can synchronize all the mismatches for the component by clicking the **Sync All** button.
3. To update the system information for all the IP components in your current system, click **Sync All System Infos** in the bottom right corner of the Qsys Pro main frame.

   Note: Clicking the update icon near the search field in the **Hierarchy** tab also synchronizes the system information for all the IP components in your system.

Figure 124. Updating System Information



9.3.10 View Your Qsys Pro System

Qsys Pro allows you to change the display of your system to match your design development. Each tab on **View** menu allows you to view your design with a unique perspective. Multiple tabs open in your workspace allows you to focus on a selected element in your system under different perspectives.

The Qsys Pro GUI supports global selection and edit. When you make a selection or apply an edit in the **Hierarchy** tab, Qsys Pro updates all other open tabs to reflect your action. For example, when you select `cpu_0` in the **Hierarchy** tab, Qsys Pro updates the **Parameters** tab to show the parameters for `cpu_0`.

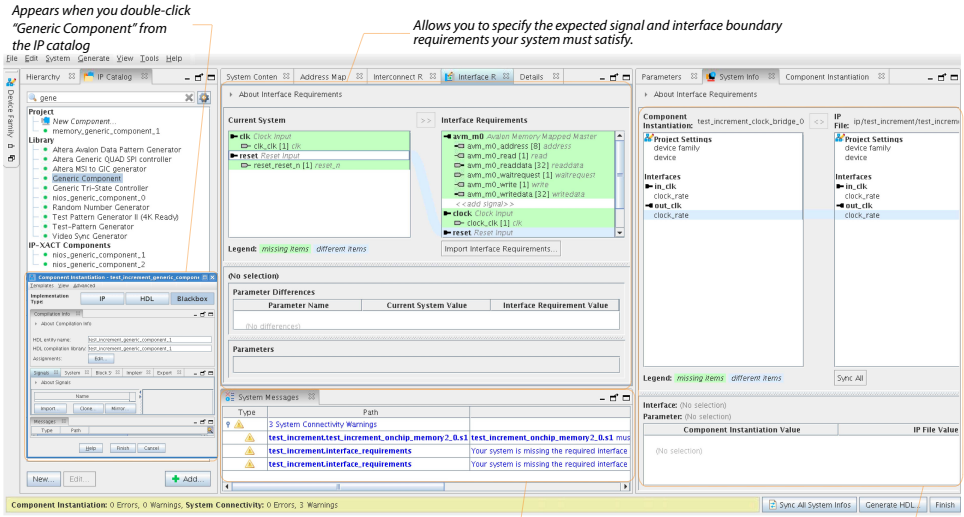
- By default, when you open Qsys Pro, the **IP Catalog**, **Hierarchy**, and the **Device Family** tabs appear to the left of the main frame.
- The **System Contents**, **Address Map**, **Interconnect Requirements**, and **Details** tabs display in the main frame.
- **Parameters**, **System Info**, and **Component Instantiation** tabs appear to the right of the main frame.
- The **System Messages** tab displays in the lower portion of Qsys Pro.
- The **Parameterization Messages** tab appears in the lower portion of the **Parameter** tab when you select an IP component, displaying parameter warnings and error messages, specific to that component.

  Note: The **Parameterization Messages** tab also appears in the bottom pane of the parameter editor when you double-click an IP component from the IP Catalog.

You can dock tabs in the main frame as a group, or individually by clicking the tab control in the upper-right corner of the main frame. You can arrange your workspace by dragging and dropping, and then grouping tabs in an order appropriate to your design development, or close or dock tabs that you are not using. Tool tips on the upper-right corner of the tab describe possible workspace arrangements, for example, restoring or disconnecting a tab to or from your workspace. When you save your system, Qsys Pro also saves the current workspace configuration. When you re-open a saved system, Qsys Pro restores the last saved workspace.

The **Reset to System Layout** command on the **View** menu restores the workspace to its default configuration for Qsys Pro system design. The **Reset to IP Layout** command restores the workspace to its default configuration for defining and generating single IP cores.

Figure 125. Qsys Pro GUI



9.3.10.1 Manage Qsys Pro Window Views with Layouts

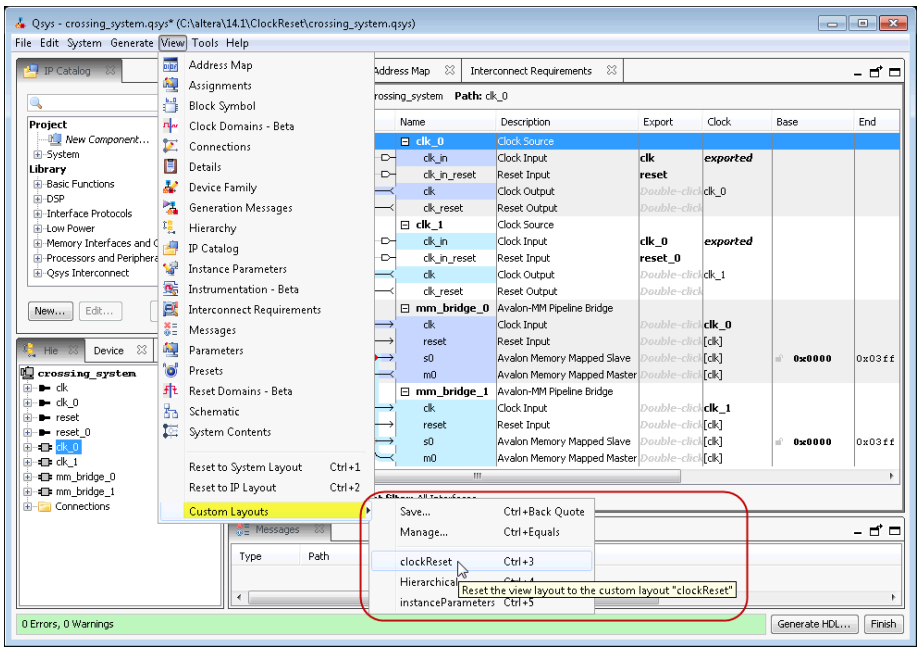
Qsys Pro Layout controls what tabs are open in your Qsys Pro design window. When you create a Qsys Pro window configuration that you want to keep, Qsys Pro allows you to save that configuration as a custom layout. The Qsys Pro GUI and features are well-suited for Qsys Pro system design. You can also use Qsys Pro to define and generate single IP cores for use in your Quartus Prime software projects.

1. To configure your Qsys Pro window with a layout suitable for Qsys Pro system design, click **View ➤ Reset to System Layout**.
   The **System Contents**, **Address Map**, **Interconnect Requirements**, and **Messages** tabs open in the main pane, and the **IP Catalog** and **Hierarchy** tabs along the left pane.
2. To configure your Qsys Pro window with a layout suitable for single IP core design, click **View ➤ Reset to IP Layout**.
   The **Parameters** and **Messages** tabs open in the main pane, and the **Details**, **Block Symbol** and **Presets** tabs along the right pane.
3. To save your current Qsys Pro window configuration as a custom layout, click **View ➤ Custom Layouts ➤ Save**.

Qsys Pro saves your custom layout in your project directory, and adds the layout to the custom layouts list, and the `layouts.ini` file. The `layouts.ini` file controls the order in which the layouts appear in the list.

4. To reset your Qsys Pro window configuration to a previously saved configuration, click **View ➤ Custom Layouts**, and then select the custom layout in the list. The Qsys Pro windows opens with your previously saved Qsys Pro window configuration.

Figure 126. Save Your Qsys Pro Window Views and Layouts

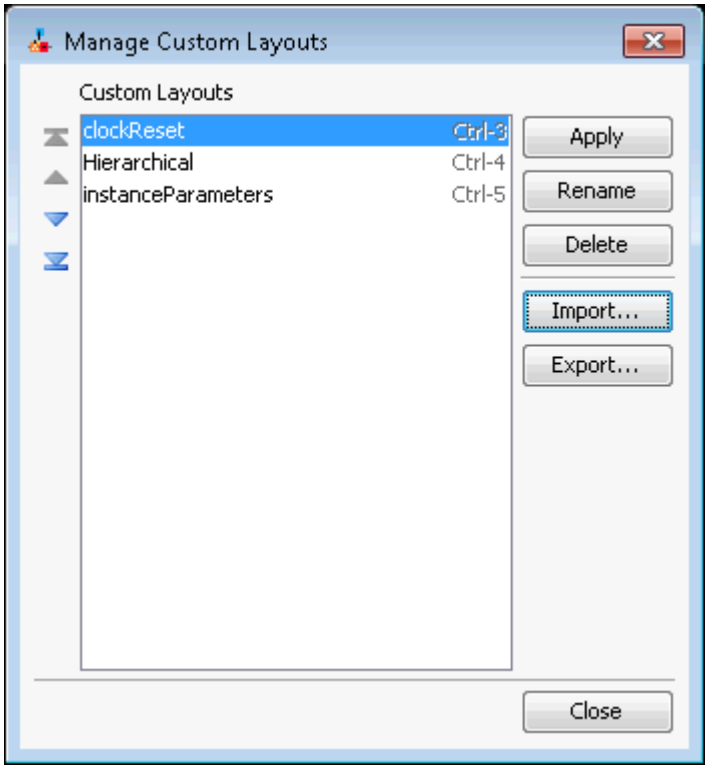


5. To manage your saved custom layouts, click **View ➤ Custom Layouts**.

   The **Manage Custom Layouts** dialog box opens and allows you to apply a variety of functions that facilitate custom layout management. For example, you can import or export a layout from or to a different directory.

Figure 127. Manage Custom Layouts

The shortcut, **Ctrl-3**, for example, allows you to quickly change your Qsys Pro window view with a quick keystroke.

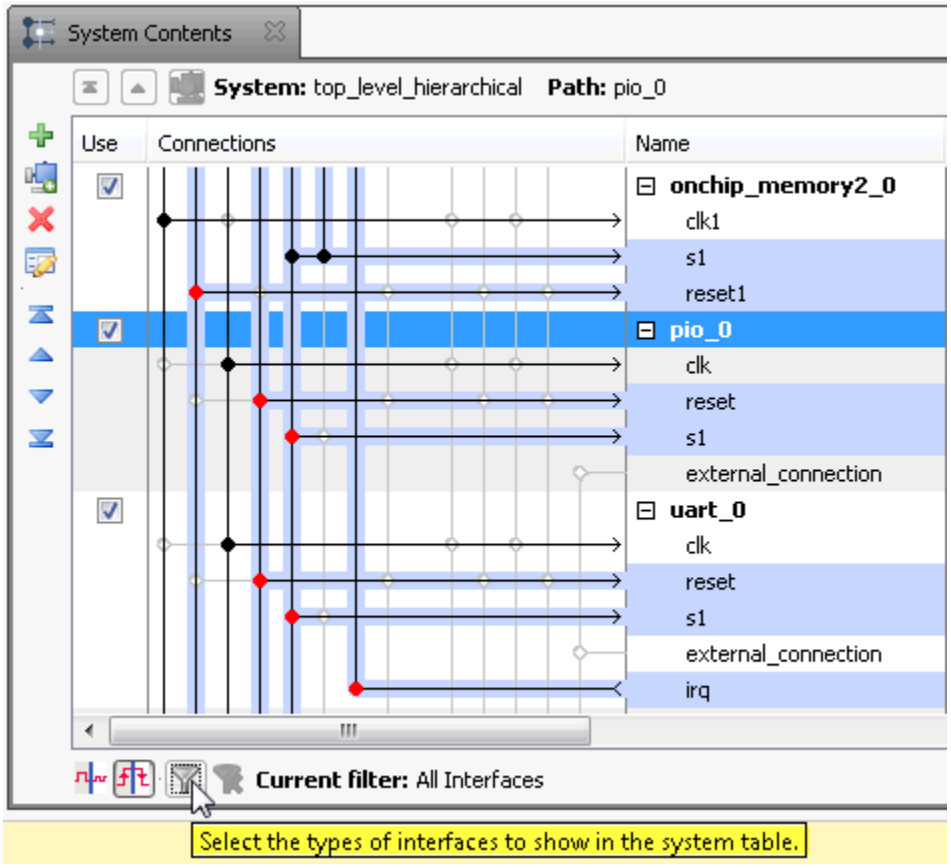


9.3.10.2 Filter the Display of the System Contents Tab

You can use the **Filters** dialog box to filter the display of your system by interface type, instance name, or by using custom tags.

For example, in the **System Contents** tab, you can show only instances that include memory-mapped interfaces or instances connected to a particular Nios II processor. The filter tool also allows you to temporarily hide clock and reset interfaces to simplify the display.

Figure 128. Filter Icon in the System Contents Tab



Related Links

Filters Dialog Box

9.3.10.3 Display Details About a Component or Parameter

The **Details** tab provides information for a selected component or parameter. Qsys Pro updates the information in the **Details** tab as you select different components.

As you click through the parameters for a component in the parameter editor, Qsys Pro displays the description of the parameter in the **Details** tab. To return to the complete description for the component, click the header in the **Parameters** tab.

9.3.10.4 Display a Graphical Representation of a Component

In the **Block Symbol** tab, Qsys Pro displays a graphical representation of the element that you select in the **Hierarchy** or **System Contents** tabs. You can view the selected component's port interfaces and signals. The **Show signals** option allows you to turn on or off signal graphics.

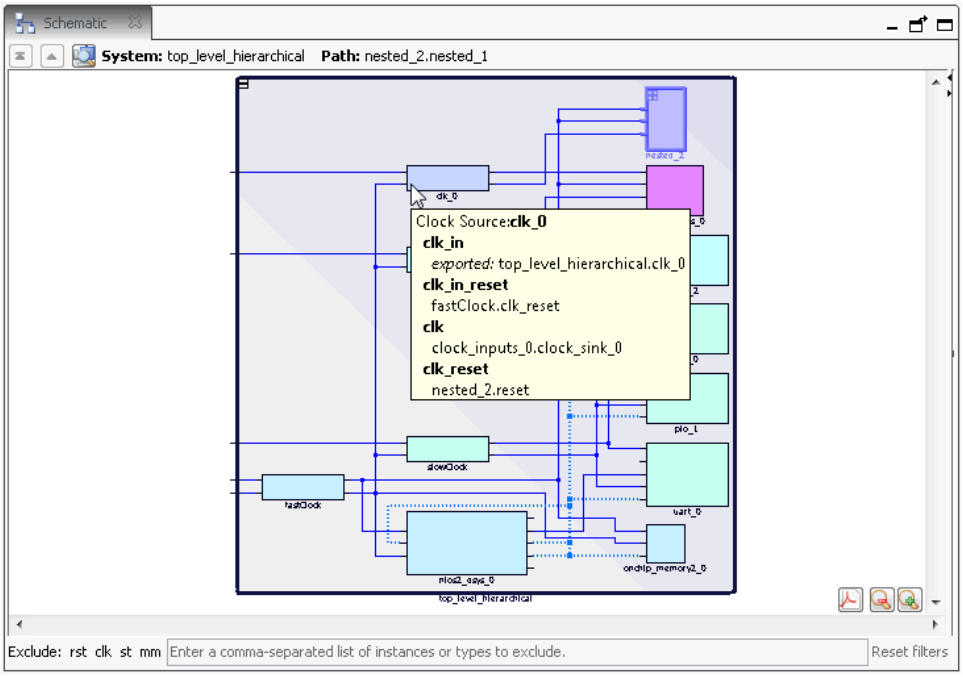
The **Block Symbol** tab appears by default in the parameter editor when you add a component to your system. When the **Block Symbol** tab is open in your workspace, it reflects changes that you make in other tabs.

9.3.10.5 View a Schematic of Your Qsys Pro System

The **Schematic** tab displays a schematic representation of your Qsys Pro system. Tab controls allow you to zoom into a component or connection, or to obtain tooltip details for your selection. You can use the image handles in the right panel to resize the schematic image.

If your selection is a subsystem, use the Hierarchy tool to navigate to the parent subsystem, move up one level, or to drill into the currently open subsystem.

Figure 129. Qsys Pro Schematic Tab



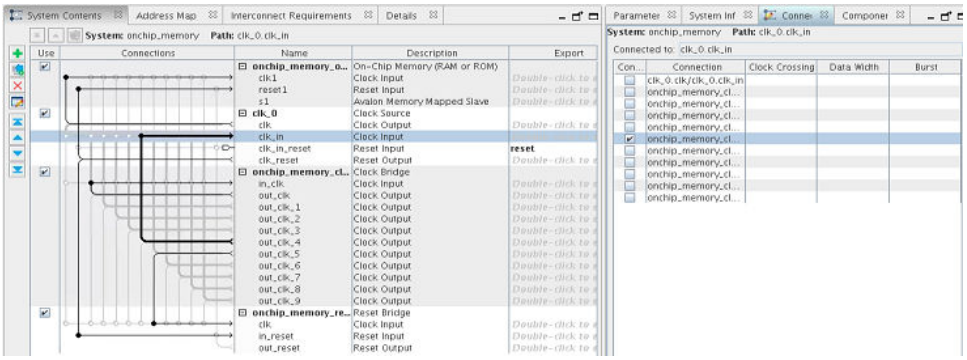
Related Links

Edit a Qsys Pro Subsystem on page 333
You can double-click a Qsys Pro subsystem in the **Hierarchy** tab to edit its contents in any tab. When you make a change, open tabs refresh their content to reflect your edit. You can change the level of a subsystem, or push it into another subsystem with commands in the **System Contents** tab.

9.3.10.6 View Connections in Your Qsys Pro System

The **Connections** tab displays a lists of connections in your Qsys Pro system. On the **Connections** tab (**View ➤ Connections**), you can choose to connect or un-connect a module in your system, and then view the results in the **System Contents** tab.

Figure 130. Connections tabs in Qsys Pro



9.3.11 Navigate Your Qsys Pro System

The **Hierarchy** tab is a full system hierarchical navigator that expands the Qsys Pro system contents to show all elements in your system.

You can use the **Hierarchy** tab to browse, connect, parameterize IP, and drive changes in other open tabs. Expanding each interface in the **Hierarchy** tab allows you to view sub-components, associated elements, and signals for the interface. You can focus on a particular area of your system by coordinating selections in the **Hierarchy** tab with other open tabs in your workspace.

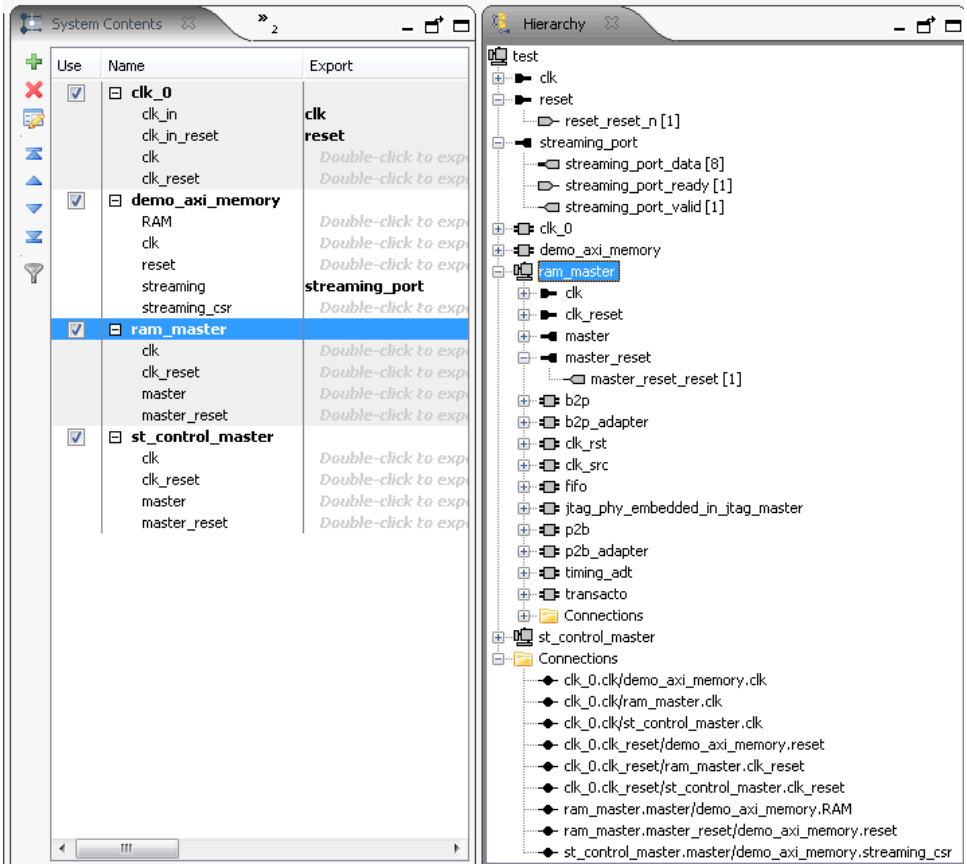
Navigating your system using the **Hierarchy** tab in conjunction with relevant tabs is useful during the debugging phase. Viewing your system with mutiple tabs open allows you to focus your debugging efforts to a single element in your system.

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

- Connections between signals.
- Names of signals in exported interfaces.
- Right-click menu to connect, edit, add, remove, or duplicate elements in the hierarchy.
- Internal connections of Qsys Pro subsystems that are included as IP components. In contrast, the **System Contents** tab displays only the exported interfaces of Qsys Pro subsystems.

Figure 131. Expanding System Contents in the Hierarchy Tab

The **Hierarchy** tab displays a unique icon for each element in the system. Context sensitivity between tabs facilitates design development and debugging. For example, when you select an element in the **Hierarchy** tab, Qsys Pro selects the same element in other open tabs. This allows you to interact with your system in more detail. In the example below, the `ram_master` selection appears selected in both the **System Contents** and **Hierarchy** tabs.



Related Links

Create and Manage Hierarchical Qsys Pro Systems on page 331
Qsys Pro supports hierarchical system design. You can add any Qsys Pro system as a subsystem in another Qsys Pro system. Qsys Pro hierarchical system design allows you to create, explore and edit hierarchies dynamically within a single instance of the Qsys Pro editor. Qsys Pro generates the complete hierarchy during the top-level system's generation.

9.3.12 Specify IP Component Parameters

The **Parameters** tab allows you to configure parameters that define an IP component's functionality.

When you add a component to your system, or when you double-click a component in an open tab, the parameter editor opens. In the parameter editor, you can configure the parameters of the component to align with the requirements of your design. If you create your own IP components, use the Hardware Component Description File (**_hw.tcl**) to specify configurable parameters.

Whenever you add an IP component to your system, Qsys Pro stores the instantiated IP component in a separate `.ip` file. Any changes you make to the component's parameters from the **Parameters** tab, automatically updates the corresponding `.ip` file.

With the **Parameters** tab open, when you select an element in the **Hierarchy** tab, Qsys Pro shows the same element in the **Parameters** tab. You can then make changes to the parameters that appear in the parameter editor, including changing the name for top-level instance that appears in the **System Contents** tab. Changes that you make in the **Parameters** tab affect your entire system and appear dynamically in other open tabs in your workspace.

In the parameter editor, the **Documentation** button provides information about a component's parameters, including the version.

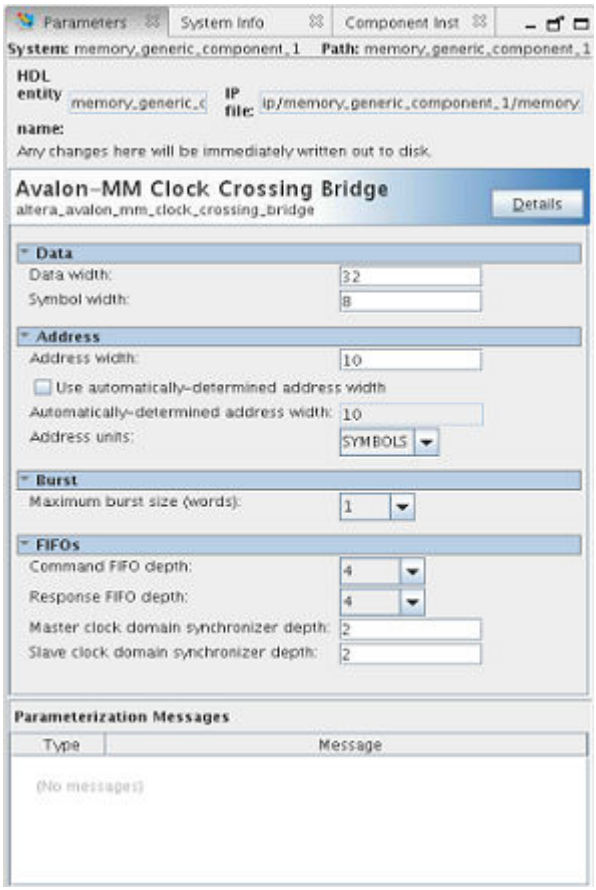
At the top of the parameter editor, Qsys Pro shows the hierarchical path for the component and its elements. This feature is useful when you navigate deep within your system with the **Hierarchy** tab.

Below the hierarchical path, the parameter editor shows the HDL entity name and the IP file path for the selected IP component.

The **Parameters** tab also allows you to review the timing for an interface and displays the read and write waveforms at the bottom of the **Parameters** tab.

The **Parameterization Messages** appears at lower portion of the parameter editor, displaying parameter warnings and error messages, specific to the selected IP component.

Figure 132. Avalon-MM Write Master Timing Waveforms in the Parameters Tab



9.3.12.1 Configure Your IP Component with a Pre-Defined Set of Parameters

The **Presets** tab allows you to apply a pre-defined set of parameters to your IP component to create a unique variation. The **Presets** tab opens the preset editor and allows you to create, modify, and save custom component parameter values as a preset file. Not all IP components have preset files.

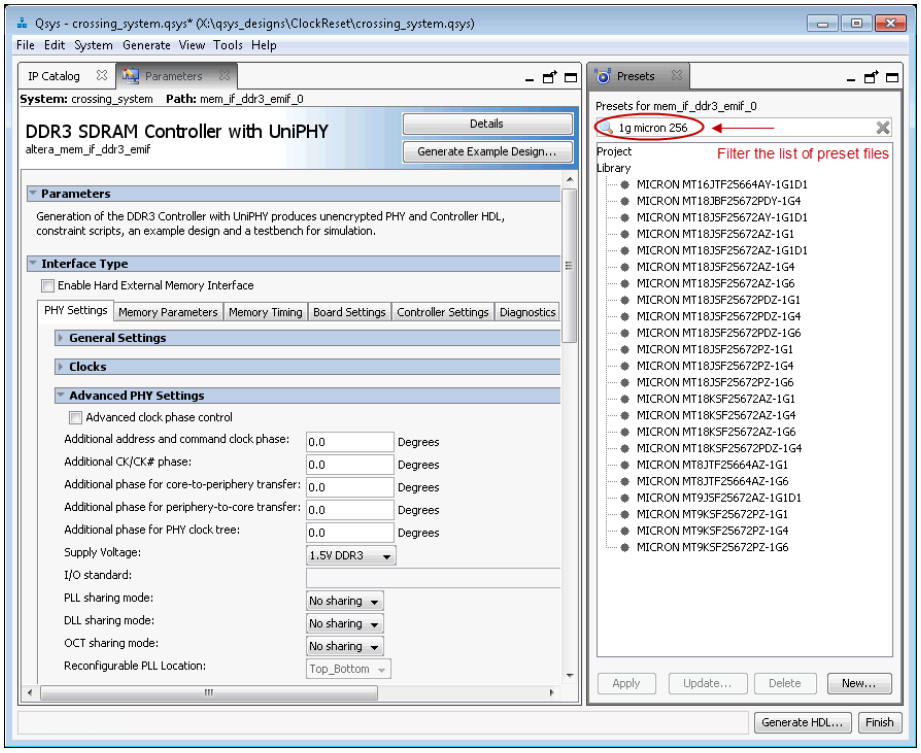
When you add a new component to your system, if there are preset files available for the component, the preset editor opens in the parameter editor. The name of each preset file describes a particular protocol.

1. In your Qsys Pro system, select an element in the **Hierarchy** tab.
2. Click **View ➤ Presets**.
3. Type text in the **Presets** search box to filter the list of preset files.
   For example, if you add the **DDR3 SDRAM Controller with UniPHY** component to your system, type `1g micron 256` in the search box, The **Presets** list displays only those preset files associated with `1g micron 256`.
4. Click **Apply** to assign the selected presets to the component.
   Presets whose parameter values match the current parameter settings appear in bold.

5. In the **Presets** tab, click **New** to create a custom preset file if the available presets do not meet the requirements of your design.
   a. In the **New Preset** dialog box, specify the **Preset name** and **Preset description**.
   b. Check or uncheck the parameters you want to include in the preset file.
   c. Specify where you want to save the new preset file.
   If the file location that you specify is not already in the IP search path, Qsys Pro adds the location of the new preset file to the IP search path.
   d. Click **Save**.
6. In the **Presets** tab, click **Update** to update a custom preset.

   Note: Custom presets are preset files that you create by clicking **New** in the **Presets** tab.
7. In the **Presets** tab, click **Delete** to delete a custom preset.

Figure 133. Specifying Presets



9.3.13 Modify an Instantiated IP Component

Qsys Pro allows you to manipulate the system representation of IP components. For example, you can modify the interfaces of an instantiated IP component to change its properties.

The example below shows how to instantiate a PLL in your system and then modify its conduit interface so that the conduit becomes a reset.

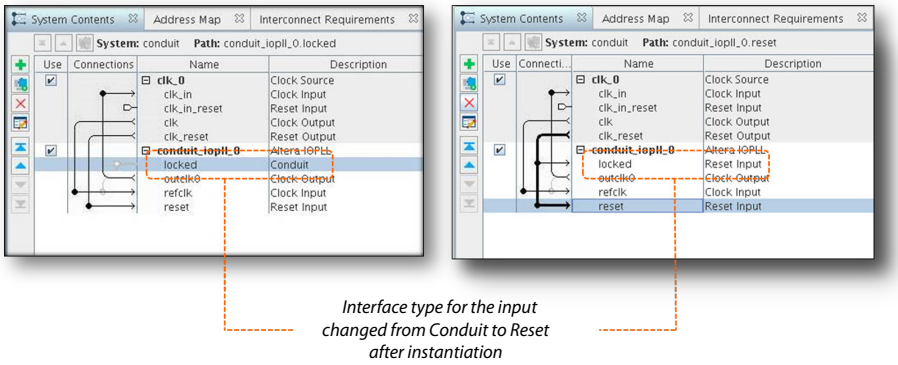
9.3.13.1 Change a Conduit to a Reset

1. In the IP Catalog search box, locate **Altera IOPLL** and double-click to add the component to your system.
2. Select the **PLL** component in the **System Contents** tab.
3. Open the **Component Instantiation** tab for the selected component.

   Note: The **Component Instantiation** tab displays in the right pane of the Qsys Pro window. If you can't find the tab on the main frame of Qsys Pro, click **View ➤ Component Instantiation** to open the tab.
4. In the **Signals & Interfaces** tab, select the **locked** conduit interface.
5. Change the **Type** from **Conduit** to **Reset Input**, and the **Synchronous edges** from **Deassert** to **None**.
6. Select the `locked [1]` signal below the **locked** interface.
7. Change the **Signal Type** from **export** to **reset_n**. Change the **Direction** from **output** to **input**.
8. Click **Apply**.

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

Figure 134. Changing Conduit to a Reset



9.3.14 Save your System

To save your Qsys Pro system, click **File ➤ Save**. To save a standalone `.ip` file that you open in the IP Parameter Editor Pro window, click **File ➤ Save**. To create a copy of the standalone `.ip` file, click **File ➤ Save As**.

Note:

- To save a copy of the Qsys Pro system, refer to the *Archive your System* section.
- To save the system as a Qsys Pro script, click **File ➤ Export System as qsys script (.tcl)**. You can restore this system by executing the `.tcl` script from the **System Scripting** tab.

Related Links

Archive your System on page 328
Qsys Pro allows you to archive your system in a `.zip` format.

9.3.15 Archive your System

Qsys Pro allows you to archive your system in a `.zip` format. To archive your system, click **File ➤ Archive System**.

In the **Archive System** dialog box, the **Collect to common directory** option is turned on by default. This option allows Qsys Pro to collect all the `.qsys` files in the root directory of the archive, and all the `.ip` files to a single `ip` directory, while updating all the references to match. Disable this option to maintain the current directory structure for the archive.

To extract all the archived files in a given system to a specified folder, click **File ➤ Restore Archive System**. Select the source archive file, and the destination folder. Upon successful extraction, Qsys Pro automatically launches the **Open System** dialog box, with the extracted `.qsys` file and the associated `.qpf` file, preloaded.

Note: You can also archive your system using command-line options. For more information, refer to *Archive a System with `qsys-archive`* section.

Related Links

Archive a Qsys Pro System with qsys-archive on page 578
The `qsys-archive` command allows you to archive a system, extract an archived system, and retrieve information about the system's dependencies.

9.4 Synchronize IP File References

Whenever you load a system, Qsys Pro ensures that the referenced IP files in your Qsys Pro system matches the IP files list in the associated Quartus Prime project.

The **IP Synchronization Result** dialog box displays the discrepancies list whenever IP synchronization mismatches occur in your Qsys Pro system. To manually check for these mismatches, click **File ➤ Synchronize IP File References**.

Qsys Pro identifies the following types of mismatches with the IP synchronization:

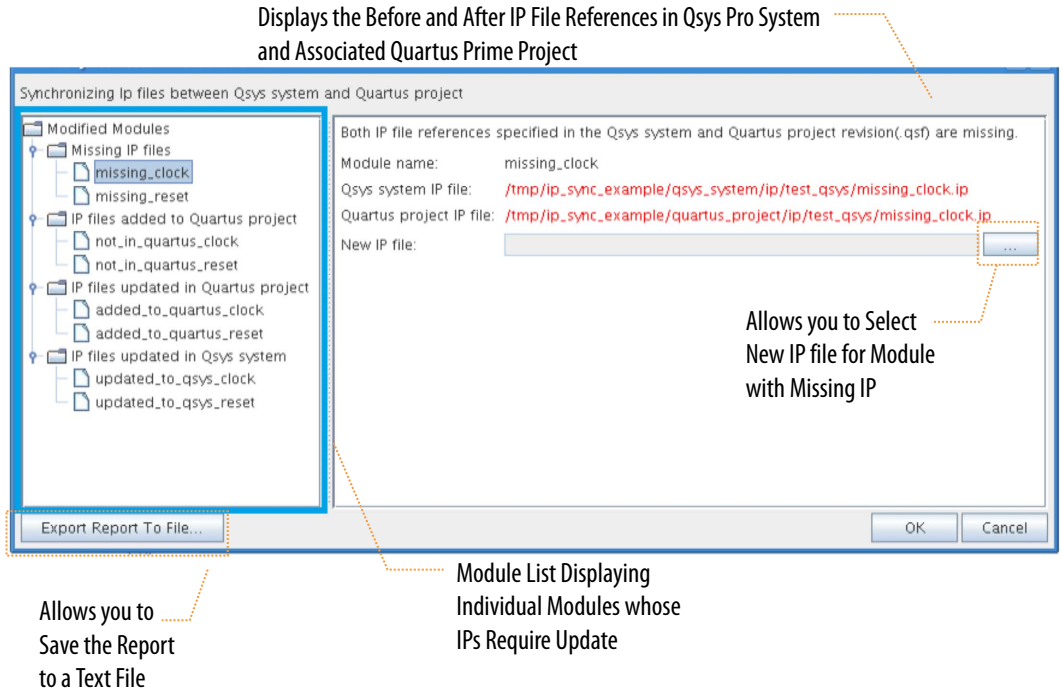
Table 87. IP Synchronization Results

Mismatch Type	Description
Duplicate IP files	The list of same IP files references specified in both your Qsys Pro system and the associated Quartus Prime project. These IP files contain the same name, but are present in different locations. In such cases, the IP files referenced in the Quartus Prime project takes precedence. Qsys Pro replaces the IP file reference in the system with the one in the Quartus Prime project.

continued...

Mismatch Type	Description
	Note: If the Quartus Prime project contains more than one IP of the same file name, Qsys Pro retains the first instance and removes all other occurrences of the IP file with the specific name.
Missing IP files	The list of missing IP file references specified in both your Qsys Pro system and the corresponding Quartus Prime project. In such cases, Qsys Pro allows you to select a replacement IP file.
Missing Qsys IP files	The list of missing IP file references in your Qsys Pro system whose associated Quartus Prime project contains valid IP files of the same names. If Qsys Pro locates a valid reference in the Quartus Prime project, it replaces the missing reference in the Qsys Pro system with IP file reference from the Quartus Prime project.
Missing Quartus IP files	The list of IP file references in your Qsys Pro system which are not listed in the associated Quartus Prime project's `.qsf` file. Qsys Pro adds the missing IP file reference to the Quartus Prime project. If the project's `.qsf` file already contains reference to the missing IP file, but the file cannot be located in the specified path, Qsys Pro removes the reference in the `.qsf` file, and adds the reference to the IP file in the Qsys Pro system.

Figure 135. Ip Synchronization Results Dialog Box



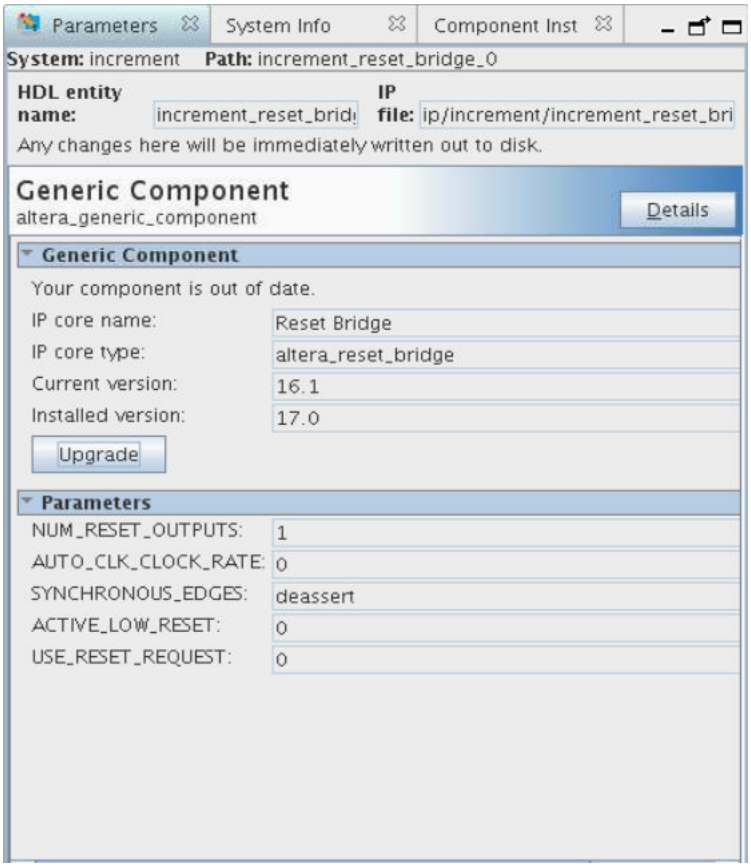
9.5 Upgrade Outdated IP Components in Qsys Pro

When you open a Qsys Pro system containing outdated IP components, you can retain and use the RTL of previously generated IP components within the Qsys Pro system. If Qsys Pro is unable to locate the IP core's original version, you cannot re-parametrize the IP core without upgrading the IP core to the latest version. However, Qsys Pro allows you to view the parametrization of the original core without upgrading.

To upgrade individual IP components in your Qsys Pro system:

1. Click **View ➤ Parameters**
2. Select the outdated IP component in the **Hierarchy** or the **System Contents** tab.
3. Click the **Parameters** tab. This tab displays information on the current version, as well as the installed version of the selected IP component.
4. Click **Upgrade**. Qsys Pro upgrades the IP component to the installed version, and deletes all the RTL files associated with the IP component.

Figure 136. Upgrade IP Component in your Qsys Pro System



To upgrade an IP component from the command-line, type the following:

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

To upgrade all the IP components in your Qsys Pro system, open the associated project in the Quartus Prime software, and click **Project ➤ Upgrade IP Components**.

Related Links

Introduction to the Qsys Pro IP Catalog on page 301
The Qsys Pro IP Catalog offers a broad range of configurable IP Cores optimized for Intel devices to use in your Qsys Pro designs.

9.6 Create and Manage Hierarchical Qsys Pro Systems

Qsys Pro supports hierarchical system design. You can add any Qsys Pro system as a subsystem in another Qsys Pro system. Qsys Pro hierarchical system design allows you to create, explore and edit hierarchies dynamically within a single instance of the Qsys Pro editor. Qsys Pro generates the complete hierarchy during the top-level system's generation.

Note: You can explore parameterizable Qsys Pro systems and `_hw.tcl` files, but you cannot edit their elements.

Your Qsys Pro systems appear in the IP Catalog under the System category under Project. You can reuse systems across multiple designs. In a team-based hierarchical design flow, you can divide large designs into subsystems and have team members develop subsystems simultaneously.

Related Links

Navigate Your Qsys Pro System on page 322
The **Hierarchy** tab is a full system hierarchical navigator that expands the Qsys Pro system contents to show all elements in your system.

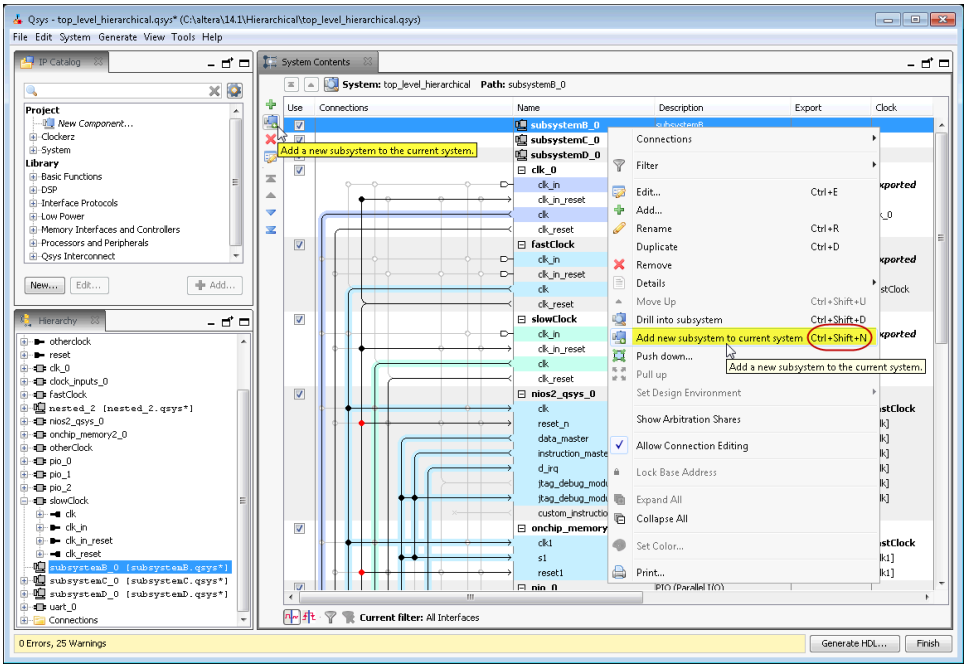
9.6.1 Add a Subsystem to Your Qsys Pro Design

You can create a child subsystem or nest subsystems at any level in the hierarchy. Qsys Pro adds a subsystem to the system you are currently editing. This can be the top-level system, or a subsystem.

To create or nest subsystems in your Qsys Pro design, use the following methods within the **System Contents** tab:

- Right-click command: **Add a new subsystem to the current system**.
- Left panel icon.
- **CTRL+SHIFT+N**.

Figure 137. Add a Subsystem to Your Qsys Pro Design



9.6.2 Drill into a Qsys Pro Subsystem to Explore its Contents

The ability to drill into a system provides visibility into its elements and connections. When you drill into an instance, you open the system it instantiates for editing.

You can drill into a subsystem with the following commands:

- Double-click a system in the **Hierarchy** tab.
- Right-click a system in the **System Contents** or **Schematic** tabs, and then select **Drill into subsystem**.
- CTRL+SHIFT+D in the **System Contents** tab.

Note: You can only drill into `.qsys` files, not parameterizable Qsys Pro systems or `_hw.tcl` files.

The **Hierarchy** tab is rooted at the top-level and drives global selection. You can manage a hierarchical Qsys Pro system that you build across multiple Qsys Pro files, and view and edit their interconnected paths and address maps simultaneously. As an example, you can select a path to a subsystem in the **Hierarchy** tab, and then drill deeper into the subsystem in the **System Contents** or **Schematic** tabs.

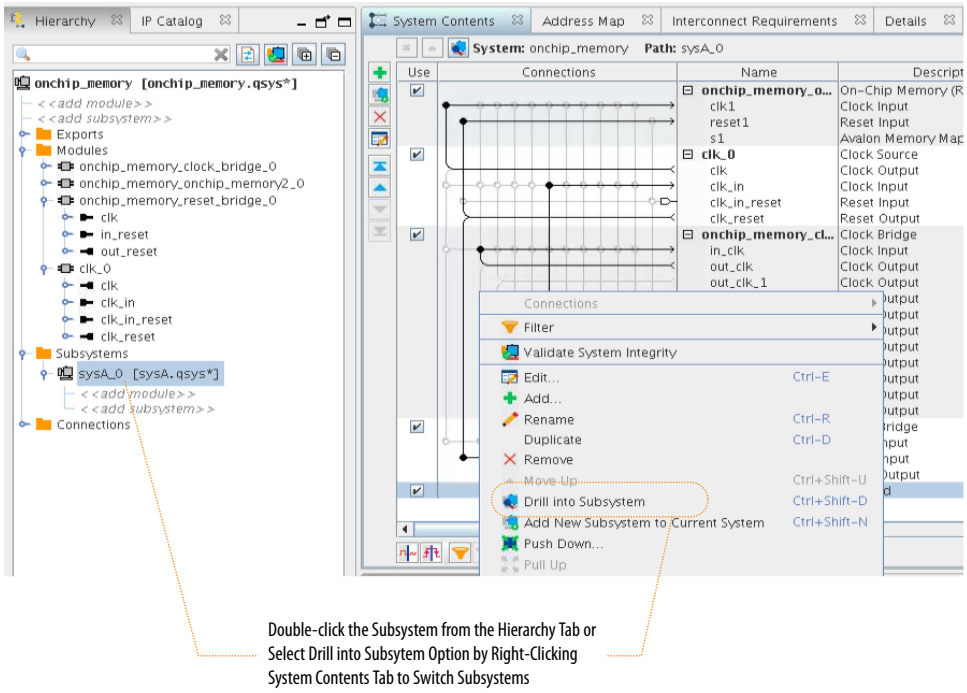
Views that manage system-level editing, for example, the **System Contents** and **Schematic** tabs, contain the hierarchy widget, which allows you to efficiently navigate your subsystems. The hierarchy widget also displays the name of the current selection, and its path in the context of the system or subsystem.

The widget contains the following controls and information:

- **Top**—Navigates to the project-level `.qsys` file that contains the subsystem.
- **Up**—Navigates up one level from the current selection.
- **Drill Into**—Allows you to drill into an editable system.
- **System**—Displays the hierarchical location of the system you are currently editing.
- **Path**—Displays the relative path to the current selection.

Note: In the **System Contents** tab, you can use CTRL+SHIFT+U to navigate up one level, and CTRL+SHIFT+D to drill into a system.

Figure 138. Drill into a Qsys Pro System to Explore its Contents



9.6.3 Edit a Qsys Pro Subsystem

You can double-click a Qsys Pro subsystem in the **Hierarchy** tab to edit its contents in any tab. When you make a change, open tabs refresh their content to reflect your edit. You can change the level of a subsystem, or push it into another subsystem with commands in the **System Contents** tab.

Note: To edit a `.qsys` file, the file must be writeable and reside outside of the ACDS installation directory. You cannot edit systems that you create from composed `_hw.tcl` files, or systems that define instance parameters.

1. In the **System Contents** or **Schematic** tabs, use the hierarchy widget to navigate to the top-level system, up one level, or down one level (drill into a system).

All tabs refresh and display the requested hierarchy level.

2. To edit a system, double-click the system in the **Hierarchy** tab. You can also drill into the system with the Hierarchy tool or right-click commands, which are available in the **Hierarchy**, **Schematic**, **System Contents** tabs.
   The system is open and available for edit in all Qsys Pro views. A system currently open for edit appears as bold in the **Hierarchy** tab.
3. In the **System Contents** tab, you can rename any element, add, remove, or duplicate connections, and export interfaces, as appropriate.
   Changes to a subsystem affect all instances. Qsys Pro identifies unsaved changes to a subsystem with an asterisk next to the subsystem in the **Hierarchy** tab.

Related Links

View a Schematic of Your Qsys Pro System on page 321
The **Schematic** tab displays a schematic representation of your Qsys Pro system. Tab controls allow you to zoom into a component or connection, or to obtain tooltip details for your selection. You can use the image handles in the right panel to resize the schematic image.

9.6.4 Change the Hierarchy Level of a Qsys Pro Component

You can push selected components down into their own subsytem, which can simplify your top-level system view. Similarly, you can pull a component up out of a subsystem to perhaps share it between two unique subsystems. Hierarchical-level management facilitates system optimization and can reduce complex connectivity in your subsystems. When you make a change, open tabs refresh their content to reflect your edit.

1. In the **System Contents** tab, to group multiple components that perhaps share a system-level component, select the components, right-click, and then select **Push down into new subsystem**.
   Qsys Pro pushes the components into their own subsystem and re-establishes the exported signals and connectivity in the new location.
2. In the **System Contents** tab, to pull a component up out of a subsystem, select the component, and then click **Pull up**.
   Qsys Pro pulls the component up out of the subsystem and re-establishes the exported signals and connectivity in the new location.

9.6.5 Save New Qsys Pro Subsystem

When you save a subsystem to your Qsys Pro design, Qsys Pro confirms the new subsystem(s) in the **Confirm New System Filenames** dialog box. The **Confirm New System Filenames** dialog box appears when you save your Qsys Pro design. Qsys Pro uses the name that you give a subsystem as `.qsys` filename, and saves the subsystems in the project's ip directory.

1. Click **File ➤ Save** to save your Qsys Pro design.
2. In the **Confirm New System Filenames** dialog box, click **OK** to accept the subsystem file names.

Note: If you have not yet saved your top-level system, or multiple subsystems, you can type a name, and then press **Enter**, to move to the next un-named system.

3. In the **Confirm New System Filenames** dialog box, to edit the name of a subsystem, click the subsystem, and then type the new name.
4. To cancel the save process, click **Cancel** in the **Confirm New System Filenames** dialog box.

9.7 Specify Signal and Interface Boundary Requirements

The **Interface Requirements** tab allows you to specify the expected signal and interface boundary requirements that your Qsys Pro system must satisfy. Use this tab to view and resolve any interface requirement mismatches in your current system. You can also edit the names of the exported signals and interfaces in your system from the **Interface Requirements** tab.

To open the **Interface Requirements** tab, click **View ➤ Interface Requirements**.

Table 88. Interface Requirements GUI Information

Name	Description
Current System	This table displays all the exported interfaces in your current Qsys Pro system. Add or remove the interfaces in the **Current System** table by adding or removing instances to the system in the **System Contents** tab.
Interface Requirements	This table shows all the interface requirements set for the current Qsys Pro system.
Parameter Differences	This table lists the **Parameter Name**, **Current System Value**, and **Interface Requirement Value** for the selected mismatched interface. *Note:* The **Interface Requirements** tab highlights in blue the signals and interfaces that are the same, but have different parameter values. Selecting a blue item populates the **Parameter Differences** table.
Import Interface Requirements	This button allows you to populate the **Interface Requirements** table from an IP-XACT file representing a generic component or an entire Qsys Pro system.
Parameters	This table lists the signal and interface parameters for the selected interface. You can view the table as **Current Parameters** when you select an interface or signal from the **Current System** table, and as **Required Parameters** when you select the signal or interface from **Interface Requirements** table. You can modify the name of your exported signal or interface from this table. For more information about how to edit the name of an exported signal or interface, refer to *Editing the Name of Exported Interfaces and Signals* in volume 1 of the *Quartus Prime Pro Edition Handbook*.

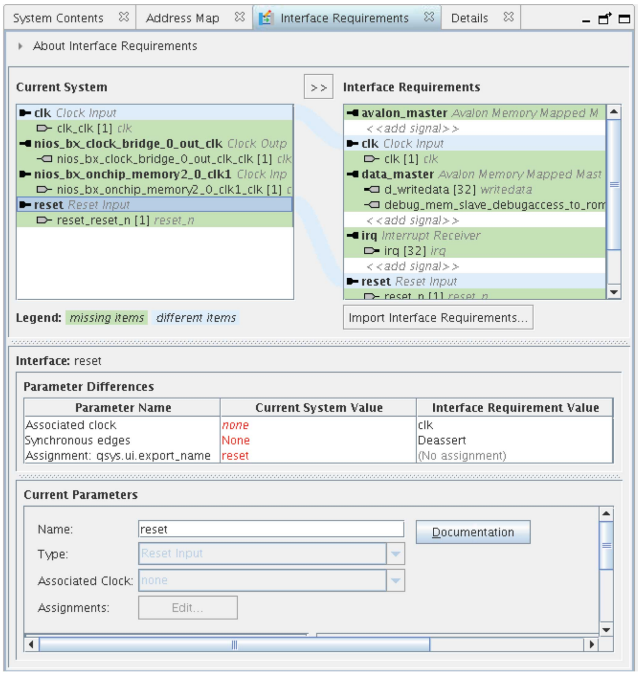
9.7.1 Match the Exported Interface with Interface Requirements

If an exported interface does not match the interface requirements of the system, Qsys Pro generates component instantiation errors. You must match all the exported interfaces with the interface requirements of the system:

1. To open the **Interface Requirements** tab, click **View ➤ Interface Requirements**.
2. To load the interface requirements from a Qsys Pro system, click **Import Interface Requirements** in the **Interface Requirements** table. A dialog box appears from which you can choose the `.ipxact` representation of the Qsys Pro system.
3. To add new interface requirements, click *<<add interface>>* or *<<add signal>>* in the **Interface Requirements** table.
4. To correct the mismatches, select the missing or mismatched interface or signal in the **Current System** table and click **>>**.

   Note: Qsys Pro highlights the mismatches between the system and interface requirements in blue, and highlights the missing interfaces and signals in green.

Figure 139. Interface Requirements Tab



Related Links

- Specify Signal and Interface Boundary Requirements on page 335
  Use this tab to view and resolve any interface requirement mismatches in your current system. You can also edit the names of the exported signals and interfaces in your system from the **Interface Requirements** tab.
- Creating System Template for a Generic Component on page 625

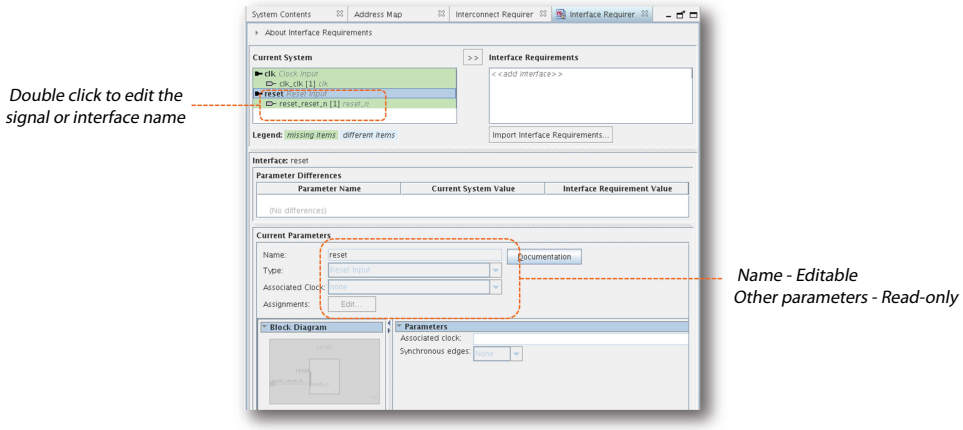
9.7.2 Edit the Name of Exported Interfaces and Signals

To rename the exported signal or interface:

- Double-click the signal or interface in **Current System** table.
- Select the signal or interface in the **Current System** table and press F2.
- Select the signal or interface in the **Current System** table and rename from the **Current Parameters** pane at the bottom of the tab. The **Current Parameters** pane displays all the parameters of the selected interface or signal.

Note: All other parameters in the **Current Parameters** except **Name** are read-only for the current system.

Figure 140. Editing the Name of Exported Interfaces and Signals



9.8 Run System Scripts

The **System Scripting** tab allows you to execute Tcl scripts on your Qsys Pro system. To open the **System Scripting** tab, click **View ➤ System Scripting**.

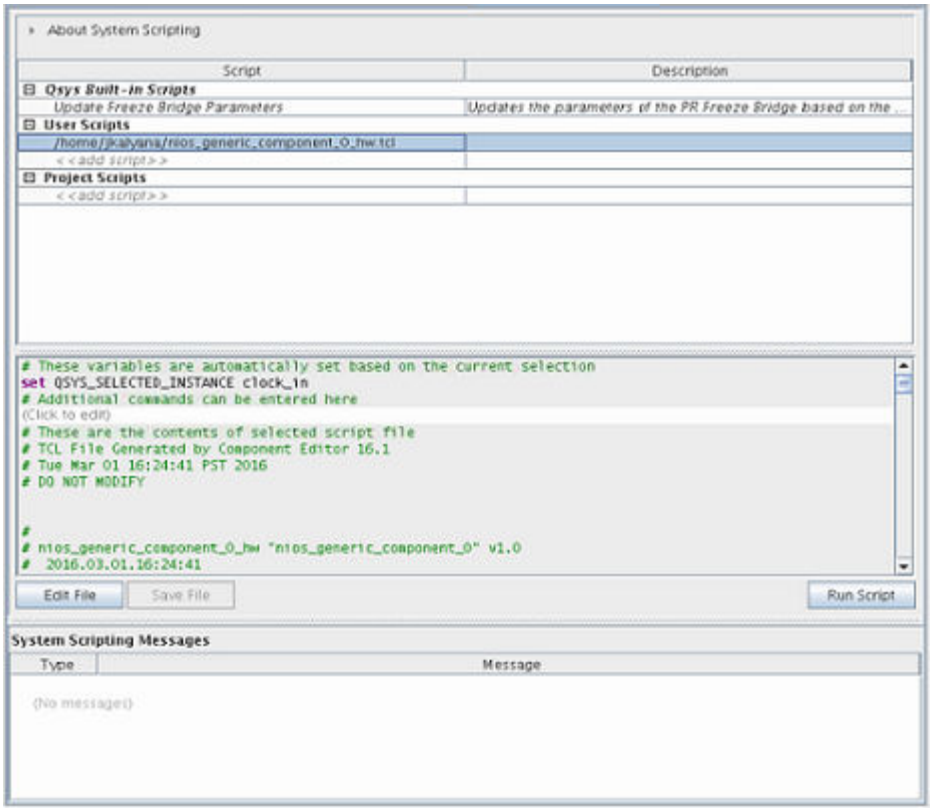
Table 89. System Scripting GUI Information

Name	Description
Qsys Built-in Scripts	Scripts that the Qsys Pro tool provides. You cannot edit these scripts.
User Scripts	You can add your own scripts to this entry. Qsys Pro saves these scripts to your user preference file, available in your home directory. The scripts that you add to this entry are available every time you open Qsys Pro. Click **<<add script>>** to add a new script file to this entry. Double-click the **Description** field to add a description. Right-click the added script and click **Rename** to set a display name for the script.
Project Scripts	You can add your own scripts to this entry. Qsys Pro saves these scripts to your current `.qsys` system. The scripts that you add to this entry are available only when you open this specific Qsys Pro system. Click **<<add script>>** to add a new script file to this entry. Double-click the **Description** field to add a description or additional commands to the script. Right-click the added script and click **Rename** to set a display name for the script.

continued...

Name	Description
Edit File	Selecting the script in the **File** field displays the script in the pane below. Click **Edit File** to edit the script.
Revert File	Discards all your changes to the edited file.
Save File	Saves your changes to the edited file.
Run Script	Executes the selected script.
System Scripting Messages	Displays the warning and error messages when running the script.

Figure 141. System Scripting Tab



Note:

- To add additional commands to run before the script, right-click the column header and enable **Additional Commands**. Selecting this option displays a third column, in addition to **File** and **Description**. Double-click the entry in this field to add commands to execute before running your script. Alternatively, you can add the additional commands to your script, directly through the display pane in the middle, in the specified section.
- You can drag and drop items between the **Project Scripts** and **User Scripts** fields.

9.9 View and Filter Clock and Reset Domains in Your Qsys Pro System

The Qsys Pro clock and reset domains tabs allow you to see clock domains and reset domains in your Qsys Pro system. Qsys Pro determines clock and reset domains by the associated clocks and resets, which are displayed in tooltips for each interface in your system. You can filter your system to display particular components or interfaces within a selected clock or reset domain. The clock and reset domain tabs also provide quick access to performance bottlenecks by indicating connection points where Qsys Pro automatically inserts clock crossing adapters and reset synchronizers during system generation. With these tools, you can more easily create optimal connections between interfaces.

Click **View ➤ Clock Domains**, or **View ➤ Reset Domains** to open the respective tabs in your workspace. The domain tools display as a tree with the current system at the root. You can select each clock or reset domain in the list to view associated interfaces.

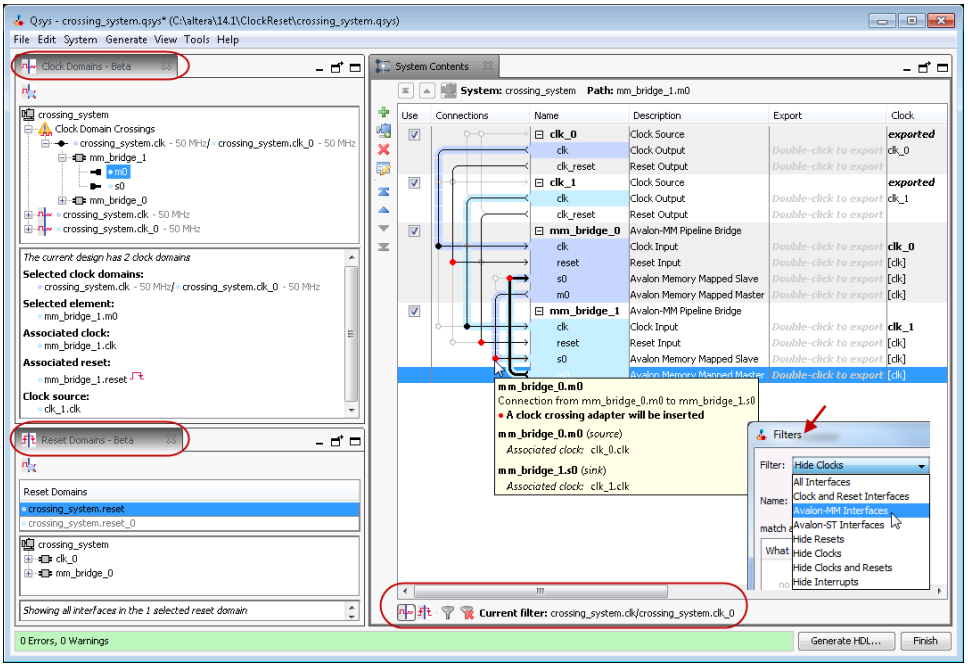
When you select an element in the **Clock Domains** tab, the corresponding selection appears in the **System Contents** tab. You can select single or multiple interface(s) and module(s). Mouse over tooltips in the **System Contents** tab to provide detailed information for all elements and connections. Colors that appear for the clocks and resets in the domain tools correspond to the colors in the **System Contents** and **Schematic** tabs.

Clock and reset control tools at the bottom on the **System Contents** tab allow you to toggle between highlighting clock or reset domains. You can further filter your view with options in the **Filters** dialog box, which is accessible by clicking the filter icon at the bottom of the **System Contents** tab. In the **Filters** dialog box, you can choose to view a single interface, or to hide clock, reset, or interrupt interfaces.

Clock and reset domain tools respond to global selection and edits, and help to provide answers to the following system design questions:

- How many clock and reset domains do you have in your Qsys Pro system?
- What interfaces and modules does each clock or reset domain contain?
- Where do clock or reset crossings occur?
- At what connection points does Qsys Pro automatically insert clock or reset adapters?
- Where do you have to manually insert a clock or reset adapter?

Figure 142. Qsys Pro Clock and Reset Domains

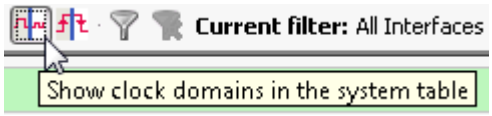


9.9.1 View Clock Domains in Your Qsys Pro System

With the **Clock Domains** tab, you can filter the **System Contents** tab to display a single clock domain, or multiple clock domains. You can further filter your view with selections in the **Filters** dialog box. When you select an element in the **Clock Domains** tab, the corresponding selection appears highlighted in the **System Contents** tab.

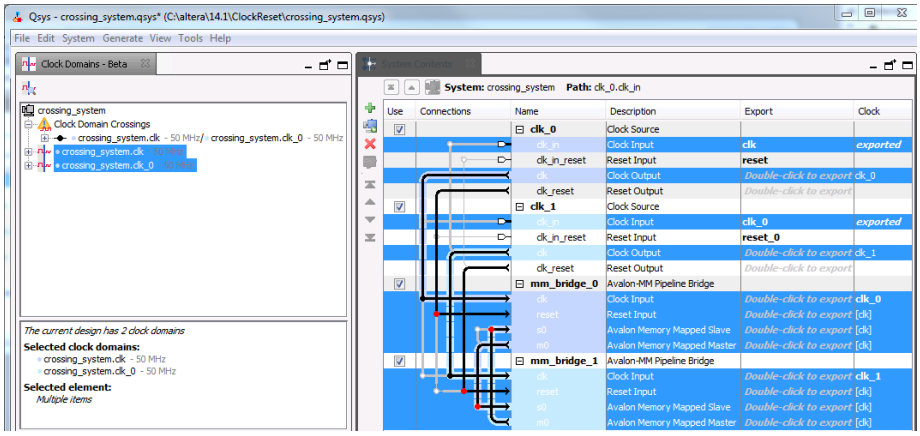
1. To view clock domain interfaces and their connections in your Qsys Pro system, click **View ➤ Clock Domains** to open the Clock Domains tab.
2. To enables and disable highlighting of the clock domains in the **System Contents** tab, click the clock control tool at the bottom of the **System Contents** tab.

Figure 143. Clock Control Tool



3. To view a single clock domain, or multiple clock domains and their modules and connections, click the clock name(s) in the **Clock Domains** tab.
   The modules for the selected clock domain(s) and their connections appear highlighted in the **System Contents** tab. Detailed information for the current selection appears in the clock domain details pane. Red dots in the **Connections** column indicate auto insertions by Qsys Pro during system generation, for example, a reset synchronizer or clock crossing adapter.

Figure 144. Clock Domains

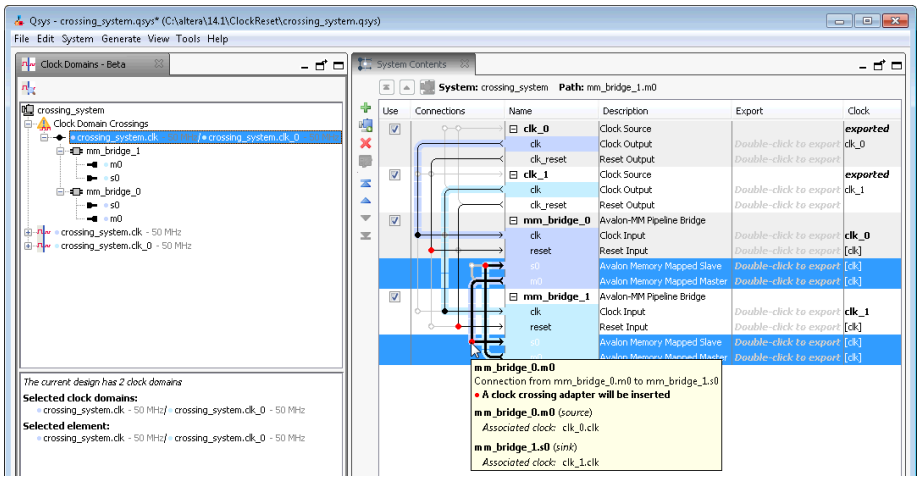


4. To view interfaces that cross clock domains, expand the **Clock Domain Crossings** icon in the **Clock Domains** tab, and select each element to view its details in the **System Contents** tab.

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

   If a connection crosses a clock domain, the connection circle appears as a red dot in the **System Contents** tab. Mouse over tooltips at the red dot connections provide details about the connection, as well as what adapter type Qsys Pro automatically inserts during system generation.

Figure 145. Clock Domain Crossings

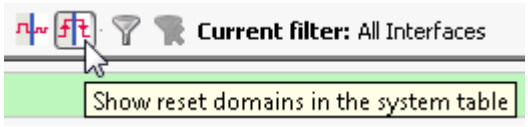


9.9.2 View Reset Domains in Your Qsys Pro System

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

1. To view reset domain interfaces and their connections in your Qsys Pro system, click **View ➤ Reset Domains** to open the **Reset Domains** tab.
2. To show reset domains in the **System Contents** tab, click the reset control tool at the bottom of the **System Contents** tab.

Figure 146. Reset Control Tool



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

   Qsys Pro displays your selection according to the following rules:

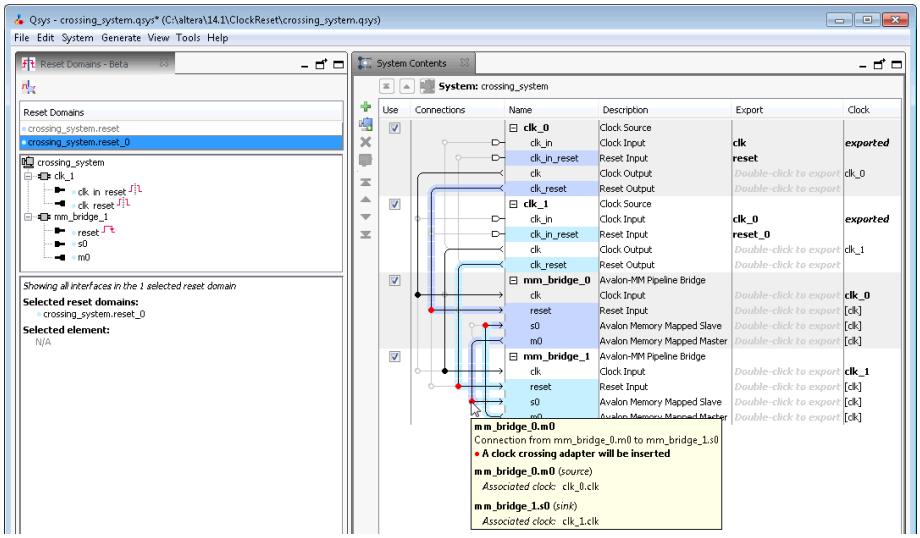
   - When you select multiple reset domains, the **System Contents** tab shows interfaces and modules in both reset domains.
   - When you select a single reset domain, the other reset domain(s) are grayed out, unless the two domains have interfaces in common.
   - Reset interfaces appear black when connected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

   Detailed information for your selection appears in the reset domain details pane.

   Note: Red dots in the **Connections** column between reset sinks and sources indicate auto insertions by Qsys Pro during system generation, for example, a reset synchronizer. Qsys Pro decides when to display a red dot with the following protocol, and ends the decision process at first match.

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

Figure 147. Reset Domains

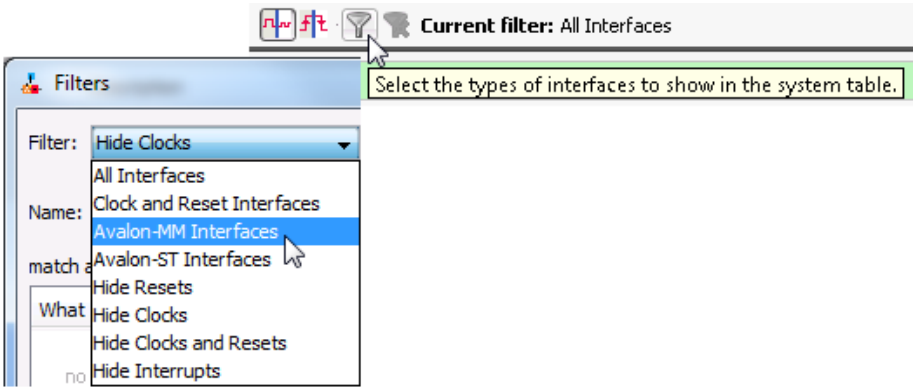


9.9.3 Filter Qsys Pro Clock and Reset Domains in the System Contents Tab

You can filter the display of your Qsys Pro clock and reset domains in the **System Contents** tab.

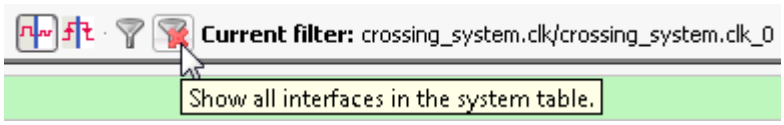
1. To filter the display in the **System Contents** tab to view only a particular interface and its connections, or to choose to hide clock, reset, or interrupt interfaces, click the **Filters** icon in the clock and reset control tool to open the **Filters** dialog box.
   The selected interfaces appear in the **System Contents** tab.

Figure 148. Filters Dialog Box



2. To clear all clock and reset filters in the **System Contents** tab and show all interfaces, click the **Filters** icon with the red "x" in the clock and reset control tool.

Figure 149. Show All Interfaces



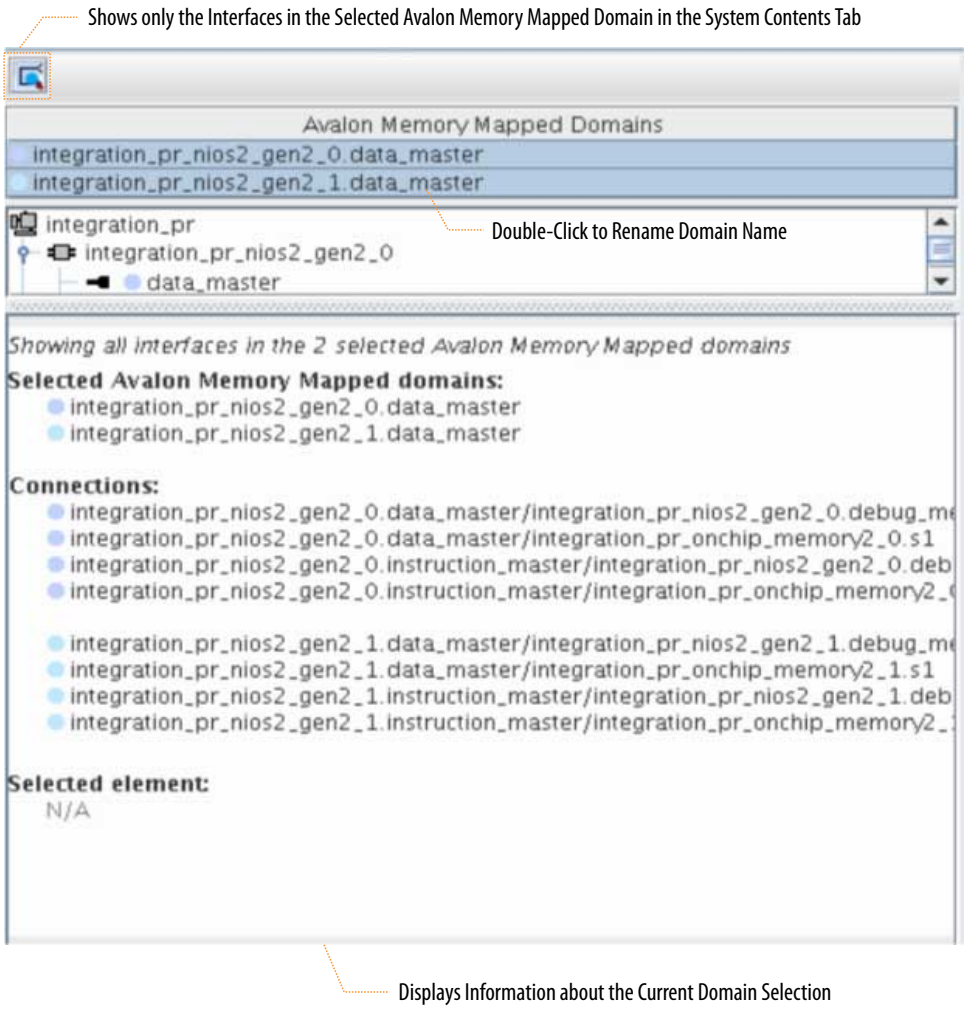
9.9.4 View Avalon Memory Mapped Domains in Your Qsys Pro System

The **Avalon Memory Mapped Domains** tab (**View ➤ Avalon Memory Mapped Domains**) displays a list of all the Avalon domains in the system.

With the **Avalon Memory Mapped Domains** tab, you can filter the **System Contents** tab to display a single Avalon domain, or multiple domains. You can further filter your view with selections in the **Filters** dialog box. When you select a domain in the **Avalon Memory Mapped Domains** tab, the corresponding selection is highlighted in the **System Contents** tab.

To rename an Avalon memory mapped domain, double-click the domain name. Detailed information for the current selection appears in the Avalon domain details pane. Also, you can choose to view only the selected domain's interfaces in the **System Contents** tab.

Figure 150. Avalon Memory Mapped Domains Tab



To enable and disable the highlighting of the Avalon domains in the **System Contents** tab, click the domain control tool at the bottom of the **System Contents** tab.

Figure 151. Avalon Memory Mapped Domains Control Tool

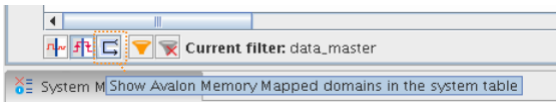
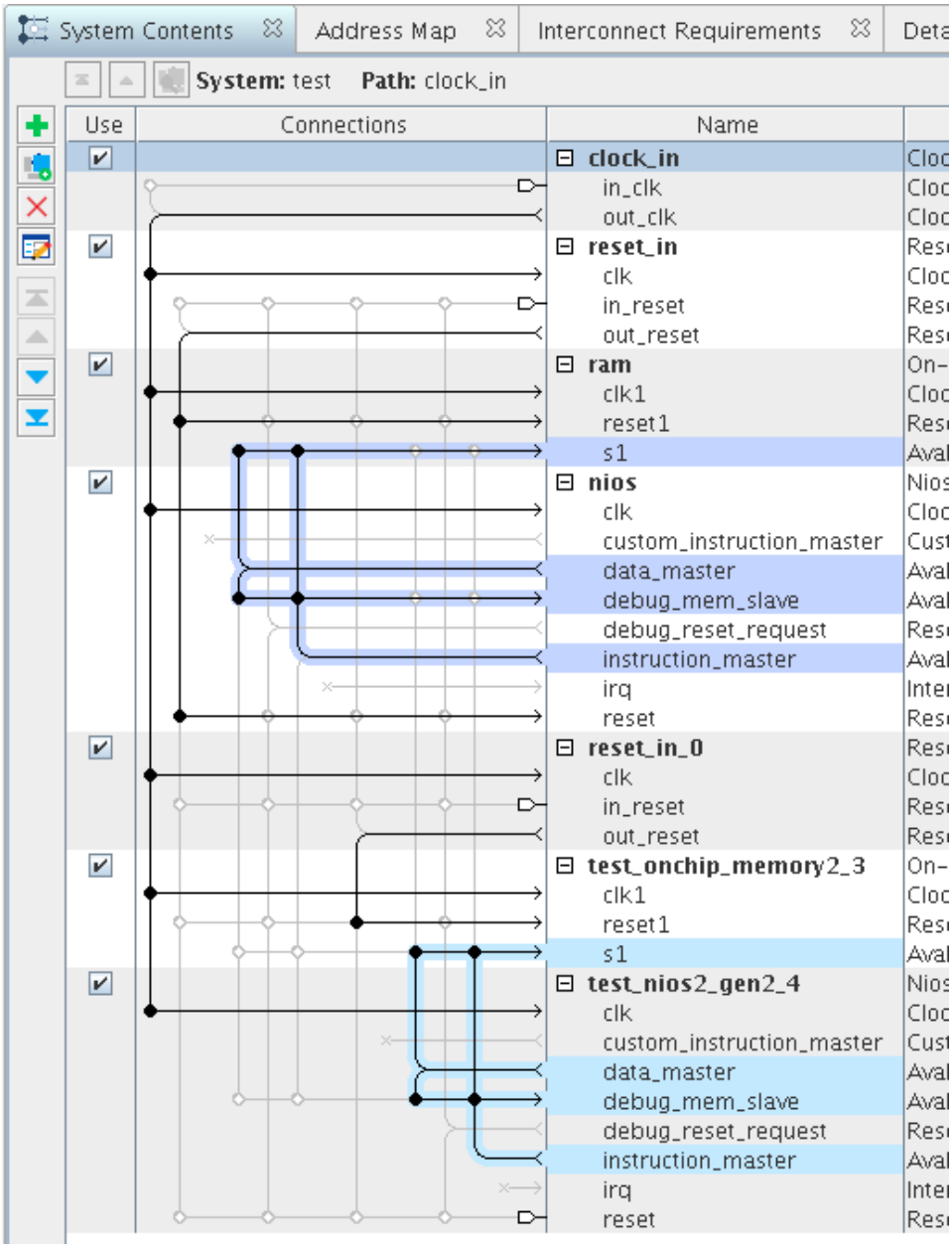


Figure 152. System Contents Tab with Avalon Memory Mapped Domains Selected



9.10 Specify Qsys Pro Interconnect Requirements

The **Interconnect Requirements** tab allows you to apply system-wide, `$system`, and interface interconnect requirements for IP components in your system. Options in the **Setting** column vary depending on what you select in the **Identifier** column. Click the drop-down menu to select the settings, and to assign the corresponding values to the settings.

Table 90. Specifying System-Wide Interconnect Requirements

Option	Description
Limit interconnect pipeline stages to	Specifies the maximum number of pipeline stages that Qsys Pro may insert in each command and response path to increase the f_{MAX} at the expense of additional latency. You can specify between 0–4 pipeline stages, where 0 means that the interconnect has a combinational datapath. Choosing 3 or 4 pipeline stages may significantly increase the logic utilization of the system. This setting is specific for each Qsys Pro system or subsystem, meaning that each subsystem can have a different setting. Additional latency is added once on the command path, and once on the response path. You can manually adjust this setting in the **Memory-Mapped Interconnect** tab. Access this tab by clicking **Show System With Qsys Pro Interconnect command** on the **System** menu.
Clock crossing adapter type	Specifies the default implementation for automatically inserted clock crossing adapters: • **Handshake**—This adapter uses a simple hand-shaking protocol to propagate transfer control signals and responses across the clock boundary. This methodology uses fewer hardware resources because each transfer is safely propagated to the target domain before the next transfer can begin. The **Handshake** adapter is appropriate for systems with low throughput requirements. • **FIFO**—This adapter uses dual-clock FIFOs for synchronization. The latency of the FIFO-based adapter is a couple of clock cycles more than the handshaking clock crossing component. However, the FIFO-based adapter can sustain higher throughput because it supports multiple transactions at any given time. FIFO-based clock crossing adapters require more resources. The **FIFO** adapter is appropriate for memory-mapped transfers requiring high throughput across clock domains. • **Auto**—If you select **Auto**, Qsys Pro specifies the **FIFO** adapter for bursting links, and the **Handshake** adapter for all other links.
Automate default slave insertion	Specifies whether you want Qsys Pro to automatically insert a default slave for undefined memory region accesses during system generation.
Enable instrumentation	When you set this option to TRUE, Qsys Pro enables debug instrumentation in the Qsys Pro interconnect, which then monitors interconnect performance in the system console.
Burst Adapter Implementation	Allows you to choose the converter type that Qsys Pro applies to each burst. • **Generic converter (slower, lower area)**—Default. Controls all burst conversions with a single converter that is able to adapt incoming burst types. This results in an adapter that has lower f_{MAX}, but smaller area. • **Per-burst-type converter (faster, higher area)**—Controls incoming bursts with a particular converter, depending on the burst type. This results in an adapter that has higher f_{MAX}, but higher area. This setting is useful when you have AXI masters or slaves and you want a higher f_{MAX}.
Enable ECC protection	Specifies the default implementation for ECC protection for memory elements. Currently supports only Read Data FIFO (`rdata_FIFO`) instances..

continued...

Option	Description
	• **FALSE**—Default. ECC protection is disabled for memory elements in the Qsys Pro interconnect. • **TRUE**—ECC protection is enabled for memory elements. Qsys Pro interconnect sends ECC errors that cannot be corrected as DECODEERROR (DECERR) on the Avalon response bus. This setting may increase logic utilization and cause lower f_{MAX}, but provides additional protection against data corruption. *Note:* For more information about Error Correction Coding (ECC), refer to *Error Correction Coding in Qsys Pro Interconnect*.

Table 91. Specifying Interface Interconnect Requirements

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

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

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

Related Links

Error Correction Coding in Qsys Pro Interconnect

9.11 Manage Qsys Pro System Security

TrustZone is the security extension of the ARM®-based architecture. It includes secure and non-secure transactions designations, and a protocol for processing between the designations. TrustZone security support is a part of the Qsys Pro interconnect.

The AXI AxPROT protection signal specifies a secure or non-secure transaction. When an AXI master sends a command, the AxPROT signal specifies whether the command is secure or non-secure. When an AXI slave receives a command, the AxPROT signal determines whether the command is secure or non-secure. Determining the security of a transaction while sending or receiving a transaction is a run-time protocol.

The Avalon specification does not include a protection signal as part of its specification. When an Avalon master sends a command, it has no embedded security and Qsys Pro recognizes the command as non-secure. When an Avalon slave receives a command, it also has no embedded security, and the slave always accepts the command and responds.

AXI masters and slaves can be TrustZone-aware. All other master and slave interfaces, such as Avalon-MM interfaces, are non-TrustZone-aware. You can set compile-time security support for all components (except AXI masters, including AXI3, AXI4,and AXI4-Lite) in the **Security** column in the **System Contents** tab, or in the **Interconnect Requirements** tab under the **Identifier** column for the master or slave interface. To begin creating a secure system, you must first add masters and

slaves to your system, and the connections between them. After you establish connections between the masters and slaves, you can then set the security options, as needed

An example of when you may need to specify compile-time security support is when an Avalon master needs to communicate with a secure AXI slave, and you can specify whether the connection point is secure or non-secure. You can specify a compile-time secure address ranges for a memory slave if an interface-level security setting is not sufficient.

Related Links

- Qsys Pro Interconnect on page 627
  Qsys Pro interconnect is a high-bandwidth structure that allows you to connect IP components to other IP components with various interfaces.
- Qsys Pro System Design Components on page 879
  You can use Qsys Pro IP components to create Qsys Pro systems.

9.11.1 Configure Qsys Pro Security Settings Between Interfaces

The AXI AxPR0T signal specifies a transaction as secure or non-secure at runtime when a master sends a transaction. Qsys Pro identifies AXI master interfaces as TrustZone-aware. You can configure AXI slaves as Trustzone-aware, secure, non-secure, or secure ranges.

Table 92. Compile-Time Security Options

For non-TrustZone-aware components, compile-time security support options are available in Qsys Pro on the **System Contents** tab, or on the **Interconnect Requirements** tab.

Compile-Time Security Options	Description
Non-secure	Master sends only non-secure transactions, and the slave receives any transaction, secure or non-secure.
Secure	Master sends only secure transactions, and the slave receives only secure transactions.
Secure ranges	Applies to only the slave interface. The specified address ranges within the slave's address span are secure, all other address ranges are not. The format is a comma-separated list of inclusive-low and inclusive-high addresses, for example, `0x0:0xfff,0x2000:0x20ff`.

After setting compile-time security options for non-TrustZone-aware master and slave interfaces, you must identify those masters that require a default slave before generation. To designate a slave interface as the default slave, turn on **Default Slave** in the **System Contents** tab. A master can have only one default slave.

Note: The **Security** and **Default Slave** columns in the **System Contents** tab are hidden by default. Right-click the **System Contents** header to select which columns you want to display.

The following are descriptions of security support for master and slave interfaces. These description can guide you in your design decisions when you want to create secure systems that have mixed secure and non-TrustZone-aware components:

- All AXI, AXI4, and AXI4-Lite masters are TrustZone-aware.
- You can set AXI, AXI4, and AXI4-Lite slaves as Trust-Zone-aware, secure, non-secure, or secure range ranges.
- You can set non-AXI master interfaces as secure or non-secure.
- You can set non-AXI slave interfaces as secure, non-secure, or secure address ranges.

9.11.2 Specify a Default Slave in a Qsys Pro System

If a master issues "per-access" or "not allowed" transactions, your design must contain a default slave. Per-access refers to the ability of a TrustZone-aware master to allow or disallow access or transactions. A transaction that violates security is rerouted to the default slave and subsequently responds to the master with an error. You can designate any slave as the default slave.

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

You can achieve an optimized secure system by partitioning your design and carefully designating secure or non-secure address maps to maintain reliable data. Avoid a design where, under the same hierarchy, a non-secure master initiates transactions to a secure slave resulting in unsuccessful transfers.

Table 93. Secure and Non-Secure Access Between Master, Slave, and Memory Components

Transaction Type	TrustZone-aware Master	Non-TrustZone-aware Master Secure	Non-TrustZone-aware Master Non-Secure
TrustZone-aware slave/memory	OK	OK	OK
Non-TrustZone-aware slave (secure)	Per-access	OK	Not allowed
Non-TrustZone-aware slave (non-secure)	OK	OK	OK
Non-TrustZone-aware memory (secure region)	Per-access	OK	Not allowed
Non-TrustZone-aware memory (non-secure region)	OK	OK	OK

9.11.3 Access Undefined Memory Regions

When a transaction from a master targets a memory region that is not specified in the slave memory map, it is known as an "access to an undefined memory region." To ensure predictable response behavior when this occurs, you must add a default slave to your design. Qsys Pro then routes undefined memory region accesses to the default slave, which terminates the transaction with an error response.

You can designate any memory-mapped slave as a default slave. Intel recommends that you have only one default slave for each interconnect domain in your system. Accessing undefined memory regions can occur in the following cases:

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

To designate a slave as the default slave, for the selected component, turn on **Default Slave** in the **Systems Content** tab.

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

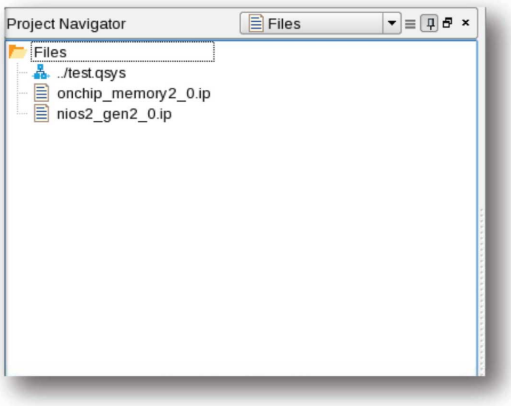
Related Links

Qsys Pro System Design Components on page 879
You can use Qsys Pro IP components to create Qsys Pro systems.

9.12 Integrating a Qsys Pro System with a Quartus Prime Project

The Quartus Prime software tightly links with Qsys Pro system creation. Qsys Pro requires you to specify a Quartus Prime project at time of system creation. The Quartus Prime software automatically adds all `.qsys` and all `.ip` files for the associated Qsys Pro system to your Quartus Prime project. When you open your Quartus Prime project, the project automatically lists all the files related to the Qsys Pro system.

Figure 153. Qsys Pro System Files in Quartus Prime Project



9.13 Manage IP Settings in the Quartus Prime Software

To specify the following IP Settings in the Quartus Prime software, click **Tools ➤ Option ➤ IP Settings**:

Table 94. IP Settings

Setting	Description
Maximum Qsys Pro memory usage	Allows you to increase memory usage for Qsys Pro if you experience slow processing for large systems, or if Qsys Pro reports an **Out of Memory** error.
IP generation HDL preference	The Quartus Prime software uses this setting when the **.qsys** file appears in the **Files** list for the current project in the **Settings** dialog box and you run Analysis & Synthesis. Qsys Pro uses this setting when you generate HDL files.
Automatically add Quartus Prime IP files to all projects	The Quartus Prime software uses this setting when you create an IP core file variation with options in the Quartus Prime IP Catalog and parameter editor. When turned on, the Quartus Prime software adds the IP variation files to the project currently open.
IP Catalog Search Locations	The Quartus Prime software uses the settings that you specify for global and project search paths under **IP Search Locations**, to populate the Quartus Prime software IP Catalog. Qsys Pro uses the settings that you specify for global search paths under **IP Search Locations** to populate the Qsys Pro IP Catalog, which appears in Qsys Pro (**Tools ➤ Options**). Qsys Pro uses the project search path settings to populate the Qsys Pro IP Catalog when you open Qsys Pro from within the Quartus Prime software (**Tools ➤ Qsys Pro)**, but not when you open Qsys Pro from the command-line.

Note: You can also access **IP Settings** by clicking **Assignments ➤ Settings ➤ IP Settings**. This access is available only when you have a Quartus Prime project open. This allows you access to **IP Settings** when you want to create IP cores independent of a Quartus Prime project. Settings that you apply or create in either location are shared.

9.13.1 Opening Qsys Pro with Additional Memory

If your Qsys Pro system requires more than the 512 megabytes of default memory, you can increase the amount of memory either in the Quartus Prime software **Options** dialog box, or at the command-line.

- When you open Qsys Pro from within the Quartus Prime software, you can increase memory for your Qsys Pro system, by clicking **Tools ➤ Options ➤ IP Settings**, and then selecting the appropriate amount of memory with the **Maximum Qsys Pro memory usage** option.
- When you open Qsys Pro from the command-line, you can add an option to increase the memory. For example, the following `qsys-edit` command allows you to open Qsys Pro with 1 gigabytes of memory.

```
qsys-edit --jvm-max-heap-size=1g
```

9.14 Generate a Qsys Pro System

In Qsys Pro, you can choose options for generation of synthesis, simulation and testbench files for your Qsys Pro system.

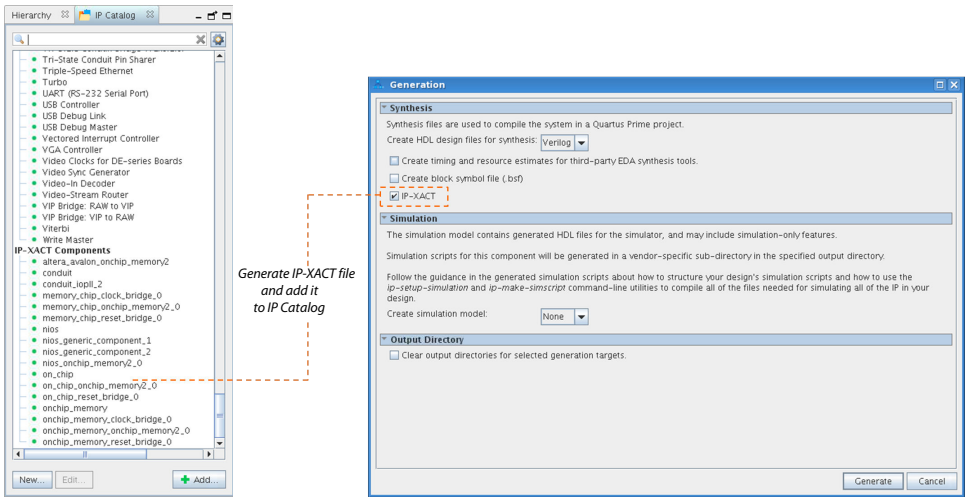
Qsys Pro system generation creates the interconnect between IP components and generates synthesis and simulation HDL files. You can generate a testbench system that adds Bus Functional Models (BFMs) that interact with your system in a simulator.

When you make changes to a system, Qsys Pro gives you the option to exit without generating. If you choose to generate your system before you exit, the **Generation** dialog box opens and allows you to select generation options.

The **Generate HDL** button in the lower-right of the Qsys Pro window allows you to quickly generate synthesis and simulation files for your system.

Note: If you cannot find the memory interface generated by Qsys Pro when you use EMIF (External Memory Interface Debug Toolkit), verify that the `.sopcinfo` file appears in your Qsys Pro project folder.

Figure 154. Generating IP-XACT file for the system



Related Links

- Avalon Verification IP Suite User Guide
- Mentor Verification IP (VIP) Altera Edition (AE)
- External Memory Interface Debug Toolkit

9.14.1 Set the Generation ID

The **Generation Id** parameter is a unique integer value that is set to a timestamp during Qsys Pro system generation. System tools, such as Nios II or HPS (Hard Processor System) use the **Generation ID** to ensure software-build compatibility with your Qsys Pro system.

To set the **Generation Id** parameter, select the top-level system in the **Hierarchy** tab, and then locating the parameter in the open **Parameters** tab.

9.14.2 Generate Files for Synthesis and Simulation

Qsys Pro generates files for synthesis in Quartus Prime software and simulation in a third-party simulator.

In Qsys Pro, you can generate simulation HDL files (**Generate ➤ Generate HDL**), which can include simulation-only features targeted towards your simulator. You can generate simulation files as Verilog or VHDL.

Note: For a list of Intel-supported simulators, refer to *Simulating Intel Designs*.

Qsys Pro supports standard and legacy device generation. Standard device generation refers to generating files for the Arria 10 and later device families. Legacy device generation refers to generating files for device families prior to the release of the Arria 10 device family, including MAX 10 devices.

The **Output Directory** option applies to both synthesis and simulation generation. By default, the path of the generation output directory is fixed relative to the `.qsys` file. You can change the default directory in the **Generation** dialog box for legacy devices. For standard devices, the generation directory is fixed to the Qsys Pro project directory.

Note: If you need to change top-level I/O pin or instance names, create a top-level HDL file that instantiates the Qsys Pro system. The Qsys Pro-generated output is then instantiated in your design without changes to the Qsys Pro-generated output files.

The following options in the **Generation** dialog box (**Generate ➤ Generate HDL**) allow you to generate synthesis and simulation files:

Table 95. Generation Dialog Box Options

Option	Description
Create HDL design files for synthesis	Generates Verilog HDL or VHDL design files for the system's top-level definition and child instances for the selected target language. Synthesis file generation is optional.
Create timing and resource estimates for third-party EDA synthesis tools	Generates a non-functional Verilog Design File (**.v**) for use by some third-party EDA synthesis tools. Estimates timing and resource usage for your IP component. The generated netlist file name is **<your_ip_component_name>_syn.v**.
Create Block Symbol File (.bsf)	Allows you to optionally create a (**.bsf**) file to use in a schematic Block Diagram File (**.bdf**).
IP-XACT	Generates an IP-XACT file for the system, and adds the file to the IP Catalog.
Create simulation model	Allows you to optionally generate Verilog HDL or VHDL simulation model files, and simulation scripts.
Clear output directories for selected generation targets	Clears previous generation attempts for current synthesis or simulation.

Note: ModelSim - Intel FPGA Edition now supports native mixed-language (VHDL/Verilog/SystemVerilog) simulation. Therefore, Intel simulation libraries may not be compatible with single language simulators. If you have a VHDL-only license, some versions of Mentor simulators may not be able to simulate IP written in Verilog. As a workaround, you can use ModelSim - Intel FPGA Edition, or purchase a mixed language simulation license from Mentor.

Related Links

Simulating Intel Designs

9.14.2.1 Files Generated for Intel FPGA IP Cores and Qsys Pro Systems

The Quartus Prime Pro Edition software generates the following output file structure for IP cores and Qsys Pro systems. The Quartus Prime Pro Edition software automatically adds the `.ip` files and the generated `.qsys` files when you open your Quartus Prime Pro project.

Figure 155. Files generated for IP cores and Qsys Pro Systems

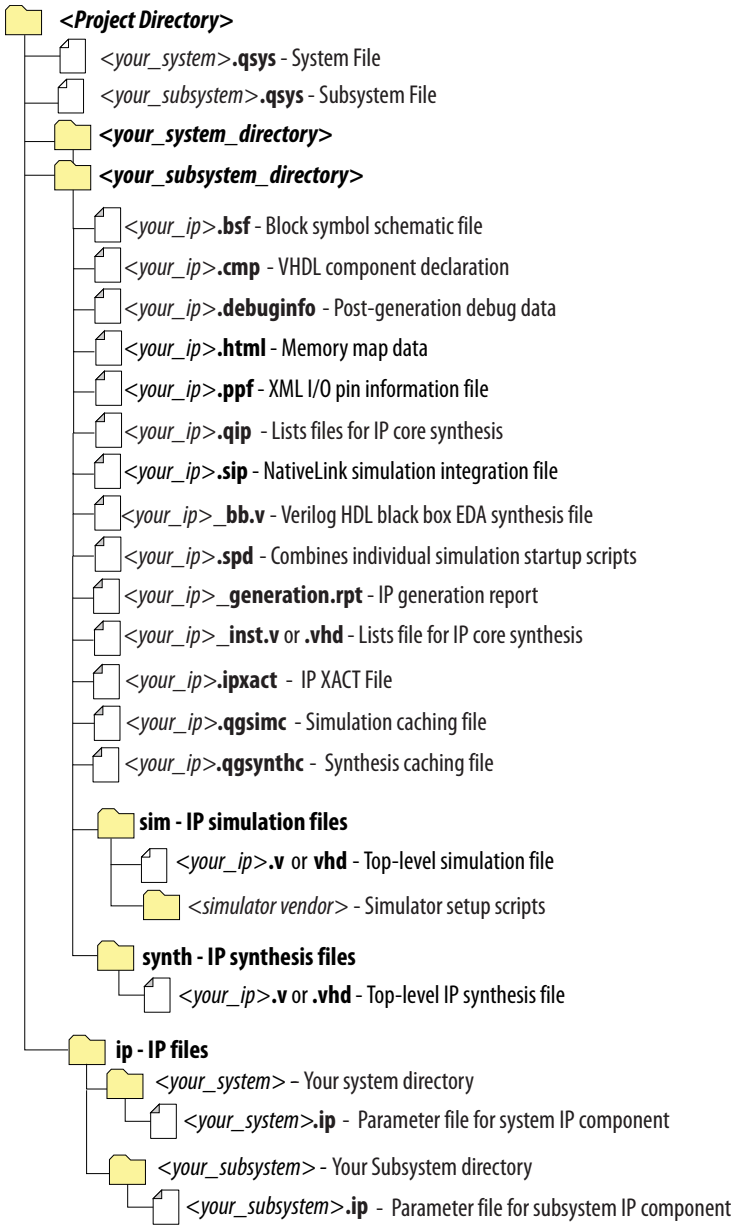


Table 96. IP Core and Qsys Pro Simulation Files

File Name	Description
<*my_system*>.qsys	The Qsys Pro system.
<*my_subsystem*>.qsys	The Qsys Pro subsystem.
ip/	Contains the parameter files for the IP components in the system and subsystem(s).
<*my_ip*>.cmp	The VHDL Component Declaration (**.cmp**) file is a text file that contains local generic and port definitions that you can use in VHDL design files.
<*my_ip*>_generation.rpt	IP or Qsys Pro generation log file. A summary of the messages during IP generation.
<*my_ip*>.qgsimc	Simulation caching file that compares the .qsys and .ip files with the current parameterization of the Qsys Pro system and IP core. This comparison determines if Qsys Pro can skip regeneration of the HDL.
<*my_ip*>.qgsynth	Synthesis caching file that compares the .qsys and .ip files with the current parameterization of the Qsys Pro system and IP core. This comparison determines if Qsys Pro can skip regeneration of the HDL.
<*my_ip*>.qip	Contains all the required information about the IP component to integrate and compile the IP component in the Quartus Prime software.
<*my_ip*>.csv	Contains information about the upgrade status of the IP component.
<*my_ip*>.bsf	A Block Symbol File (.**bsf**) representation of the IP variation for use in Block Diagram Files (**.bdf**).
<*my_ip*>.spd	Required input file for ip-make-simscript to generate simulation scripts for supported simulators. The **.spd** file contains a list of files generated for simulation, along with information about memories that you can initialize.
<*my_ip*>.ppf	The Pin Planner File (**.ppf**) stores the port and node assignments for IP components created for use with the Pin Planner.
<*my_ip*>_bb.v	Use the Verilog black box (**_bb.v**) file as an empty module declaration for use as a black box.
<*my_ip*>.sip	Contains information required for NativeLink simulation of IP components. Add the **.sip** file to your Quartus Prime Standard Edition project to enable NativeLink for supported devices. The Quartus Prime Pro Edition software does not support NativeLink simulation.
<*my_ip*>_inst.v or _inst.vhd	HDL example instantiation template. Copy and paste the contents of this file into your HDL file to instantiate the IP variation.
<*my_ip*>.regmap	If the IP contains register information, the Quartus Prime software generates the **.regmap** file. The **.regmap** file describes the register map information of master and slave interfaces. This file complements the .**sopcinfo** file by providing more detailed register information about the system. This file enables register display views and user customizable statistics in System Console.
<*my_ip*>.svd	Allows HPS System Debug tools to view the register maps of peripherals connected to HPS within a Qsys Pro system. During synthesis, the Quartus Prime software stores the .svd files for slave interface visible to the System Console masters in the .sof file in the debug session. System Console reads this section, which Qsys Pro can query for register map information. For system slaves, Qsys Pro can access the registers by name.
<*my_ip*>.v <*my_ip*>.vhd	HDL files that instantiate each submodule or child IP core for synthesis or simulation.
mentor/	Contains a ModelSim script msim_setup.tcl to set up and run a simulation.

continued...

File Name	Description
aldec/	Contains a Riviera-PRO script rivierapro_setup.tcl to setup and run a simulation.
/synopsys/vcs /synopsys/vcsmx	Contains a shell script vcs_setup.sh to set up and run a VCS® simulation. Contains a shell script vcsmx_setup.sh and synopsys_ sim.setup file to set up and run a VCS MX® simulation.
/cadence	Contains a shell script ncsim_setup.sh and other setup files to set up and run an NCSIM simulation.
/submodules	Contains HDL files for the IP core submodule.
<*IP submodule*>/	For each generated IP submodule directory, Qsys Pro generates /synth and /sim sub-directories.

9.14.3 Generate Files for a Testbench Qsys Pro System

Qsys Pro testbench is a new system that instantiates the current Qsys Pro system by adding BFMs to drive the top-level interfaces. BFMs interact with the system in the simulator. You can use options in the **Generation** dialog box (**Generate ➤ Generate Testbench System**) to generate a testbench Qsys Pro system.

You can generate a standard or simple testbench system with BFM or Mentor Verification IP (for AXI3/AXI4) IP components that drive the external interfaces of your system. Qsys Pro generates a Verilog HDL or VHDL simulation model for the testbench system to use in your simulation tool. You should first generate a testbench system, and then modify the testbench system in Qsys Pro before generating its simulation model. In most cases, you should select only one of the simulation model options.

By default, the path of the generation output directory is fixed relative to the .qsys file. You can change the default directory in the **Generation** dialog box for legacy devices. For standard devices, the generation directory is fixed to the Qsys Pro project directory.

The following options are available for generating a Qsys Pro testbench system:

Option	Description
Create testbench Qsys Pro system	• **Standard, BFMs for standard Qsys Pro Interconnect**—Creates a testbench Qsys Pro system with BFM IP components attached to exported Avalon and AXI3/AXI4 interfaces. Includes any simulation partner modules specified by IP components in the system. The testbench generator supports AXI interfaces and can connect AXI3/AXI4 interfaces to Mentor Graphics AXI3/AXI4 master/slave BFMs. However, BFMs support address widths only up to 32-bits. • **Simple, BFMs for clocks and resets**—Creates a testbench Qsys Pro system with BFM IP components driving only clock and reset interfaces. Includes any simulation partner modules specified by IP components in the system.
Create testbench simulation model	Creates Verilog HDL or VHDL simulation model files and simulation scripts for the testbench Qsys Pro system currently open in your workspace. Use this option if you do not need to modify the Qsys Pro-generated testbench before running the simulation.

Note: ModelSim - Intel FPGA Edition now supports native mixed-language (VHDL/Verilog/SystemVerilog) simulation. Therefore, Intel simulation libraries may not be compatible with single language simulators. If you have a VHDL-only license, some versions of Mentor simulators may not be able to simulate IP written in Verilog. As a workaround, you can use ModelSim - Intel FPGA Edition, or purchase a mixed language simulation license from Mentor.

9.14.3.1 Files Generated for Qsys Pro Testbench

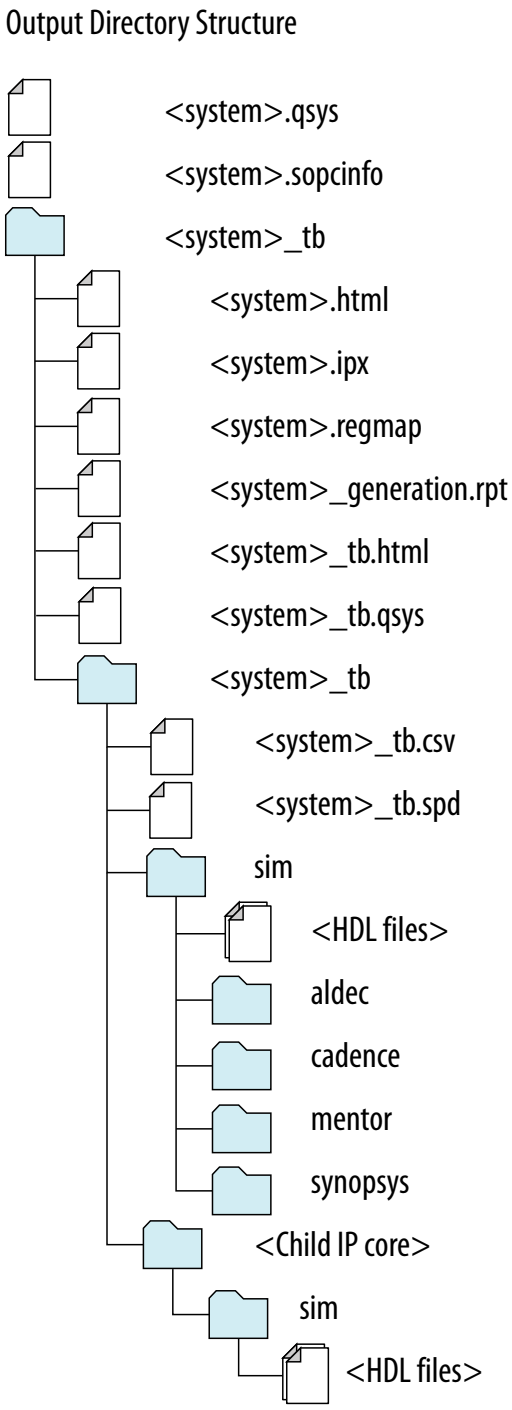
Table 97. Qsys Pro-Generated Testbench Files

File Name or Directory Name	Description
<*system*>_tb.qsys	The Qsys Pro testbench system.
<*system*>_tb.v or <*system*>_tb.vhd	The top-level testbench file that connects BFMs to the top-level interfaces of <*system*>_*tb*.qsys.
<*system*>_tb.spd	Required input file for ip-make-simscript to generate simulation scripts for supported simulators. The **.spd** file contains a list of files generated for simulation and information about memory that you can initialize.
<*system*>.html and <*system*>_tb.html	A system report that contains connection information, a memory map showing the address of each slave with respect to each master to which it is connected, and parameter assignments.
<*system*>_generation.rpt	Qsys Pro generation log file. A summary of the messages that Qsys Pro issues during testbench system generation.
<*system*>.ipx	The IP Index File (.ipx) lists the available IP components, or a reference to other directories to search for IP components.
<*system*>.svd	Allows HPS System Debug tools to view the register maps of peripherals connected to HPS within a Qsys Pro system. Similarly, during synthesis the .**svd** files for slave interfaces visible to System Console masters are stored in the **.sof** file in the debug section. System Console reads this section, which Qsys Pro can query for register map information. For system slaves, Qsys Pro can access the registers by name.
mentor/	Contains a ModelSim script msim_setup.tcl to set up and run a simulation
aldec/	Contains a Riviera-PRO script rivierapro_setup.tcl to setup and run a simulation.
/synopsys/vcs /synopsys/vcsmx	Contains a shell script vcs_setup.sh to set up and run a VCS simulation. Contains a shell script vcsmx_setup.sh and synopsys_ sim.setup file to set up and run a VCS MX® simulation.
/cadence	Contains a shell script ncsim_setup.sh and other setup files to set up and run an NCSIM simulation.
/submodules	Contains HDL files for the submodule of the Qsys Pro testbench system.
<*child IP cores*>/	For each generated child IP core directory, Qsys Pro testbench generates /synth and /sim subdirectories.

9.14.3.2 Qsys Pro Testbench Simulation Output Directories

The /sim and /simulation directories contain the Qsys Pro-generated output files to simulate your Qsys Pro testbench system.

Figure 156. Qsys Pro Simulation Testbench Directory Structure



9.14.3.3 Generate and Modify a Qsys Pro Testbench System

You can use the following steps to create a Qsys Pro testbench system of your Qsys Pro system.

1. Create a Qsys Pro system.
2. Generate a testbench system in the Qsys Pro **Generation** dialog box (**Generate ➤ Generate Testbench System**).
3. Open the testbench system in Qsys Pro. Make changes to the BFMs, as needed, such as changing the instance names and **VHDL ID** value. For example, you can modify the **VHDL ID** value in the **Altera Avalon Interrupt Source** IP component.
4. If you modify a BFM, regenerate the simulation model for the testbench system.
5. Create a custom test program for the BFMs.
6. Compile and load the Qsys Pro system and testbench into your simulator, and then run the simulation.

9.14.4 Qsys Pro Simulation Scripts

Qsys Pro generates simulation scripts to set up the simulation environment for Mentor Graphics Modelsim® and Questasim®, Synopsys VCS and VCS MX, Cadence Incisive Enterprise Simulator® (NCSIM), and the Aldec Riviera-PRO Simulator.

Qsys Pro generates simulation scripts for all `.ip` and `.qsys` files of a system and places the files in the simulation script output folder (*<top-level system name>*/**sim**/*<simulator name>*).

Qsys Pro always generates the simulation scripts from the currently loaded system down. You can open a subsystem and choose to generate a simulation script just for that subsystem.

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

Table 98. Simulation Script Variables

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

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

Example 82. Top-Level Simulation HDL File for a Testbench System

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

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

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

Note: ModelSim - Intel FPGA Edition now supports native mixed-language (VHDL/Verilog/SystemVerilog) simulation. Therefore, Intel simulation libraries may not be compatible with single language simulators. If you have a VHDL-only license, some versions of Mentor simulators may not be able to simulate IP written in Verilog. As a workaround, you can use ModelSim - Intel FPGA Edition, or purchase a mixed language simulation license from Mentor.

Related Links

Incorporating IP Simulation Scripts in Top-Level Scripts

9.14.4.1 Generating a Combined Simulator Setup Script

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

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

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

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

1. Generate, regenerate, or upgrade one or more IP core. Refer to *Generating IP Cores* or *Upgrading IP Cores*.
2. Click **Tools ➤ Generate Simulator Setup Script for IP** (or run the `ip-setup-simulation` utility). Specify the **Output Directory** and library compilation options. Click **OK** to generate the file. By default, the files generate into the */<project directory>/<simulator>/* directory using relative paths.
3. To incorporate the generated simulator setup script into your top-level simulation script, refer to the template section in the generated simulator setup script as a guide to creating a top-level script:
   a. Copy the specified template sections from the simulator-specific generated scripts and paste them into a new top-level file.
   b. Remove the comments at the beginning of each line from the copied template sections.
   c. Specify the customizations you require to match your design simulation requirements, for example:

- Specify the TOP_LEVEL_NAME variable to the design's simulation top-level file. The top-level entity of your simulation is often a testbench that instantiates your design. Then, your design instantiates IP cores and/or Qsys or Qsys Pro systems. Set the value of TOP_LEVEL_NAME to the top-level entity.
- If necessary, set the QSYS_SIMDIR variable to point to the location of the generated IP simulation files.
- Compile the top-level HDL file (e.g. a test program) and all other files in the design.
- Specify any other changes, such as using the `grep` command-line utility to search a transcript file for error signatures, or e-mail a report.

4. Re-run **Tools ➤ Generate Simulator Setup Script for IP** (or `ip-setup-simulation`) after regeneration of an IP variation.

Table 99. Simulation Script Utilities

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

The following sections provide step-by-step instructions for sourcing each simulator setup script in your top-level simulation script.

9.14.5 Simulating Software Running on a Nios II Processor

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

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

Sets up the ModelSim simulation environment, and compiles and loads the Nios II software simulation.

7. To run the simulation in ModelSim, type `run -all` in the ModelSim transcript window.
8. Set the ModelSim settings and select the Qsys Pro Testbench Simulation Package Descriptor (`.spd`) file, `< system >_tb.spd`. The `.spd` file is generated with the testbench simulation model for Nios II designs and specifies the files required for Nios II simulation.

Related Links

- Getting Started with the Graphical User Interface
  In *Nios II Gen2 Software Developer's Handbook*
- Getting Started from the Command Line
  In *Nios II Gen2 Software Developer's Handbook*

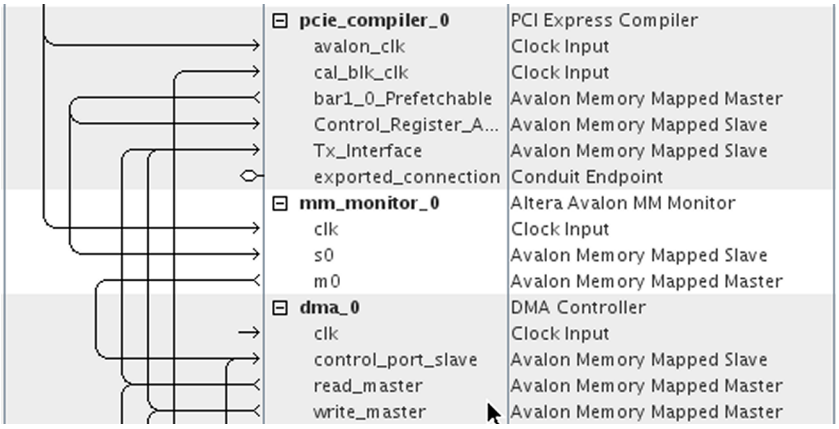
9.14.6 Add Assertion Monitors for Simulation

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

Note: ModelSim - Intel FPGA Edition does not support SystemVerilog assertions. If you want to use assertion monitors, you must use a supported third-party simulators such as Mentor Questasim, Synopsys VCS, or Cadence Incisive. For more information, refer to *Introduction to Intel FPGA IP Cores*.

Figure 157. Inserting an Avalon-MM Monitor Between an Avalon-MM Master and Slave Interface

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



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

Related Links

Introduction to Intel FPGA IP Cores

9.14.7 CMSIS Support for the HPS IP Component

Qsys Pro systems that contain an HPS IP component generate a System View Description (**.svd**) file that lists peripherals connected to the ARM processor.

The **.svd** (or CMSIS-SVD) file format is an XML schema specified as part of the Cortex Microcontroller Software Interface Standard (CMSIS) provided by ARM. The **.svd** file allows HPS system debug tools (such as the DS-5 Debugger) to view the register maps of peripherals connected to HPS in a Qsys Pro system.

Related Links

- Component Interface Tcl Reference on page 762
  Tcl commands allow you to perform a wide range of functions in Qsys Pro. Command descriptions contain the Qsys Pro phases where you can use the command, for example, main program, elaboration, composition, or fileset callback.
- CMSIS - Cortex Microcontroller Software

9.14.8 Generate Header Files

You can use the `sopc-create-header-files` command from the Nios II command shell to create header files for any master component in your Qsys Pro system. The Nios II tool chain uses this command to create the processor's `system.h` file. You can also use this command to generate system level information for a hard processing system (HPS) in Intel's SoC devices or other external processors. The header file includes address map information for each slave, relative to each master that accesses the slave. Different masters may have different address maps to access a particular slave component. By default, the header files are in C format and have a `.h` suffix. You can select other formats with appropriate command-line options.

Table 100. sopc-create-header-files Command-Line Options

Option	Description
<*sopc*>	Path to Qsys Pro `.sopcinfo` file, or the file directory. If you omit this option, the path defaults to the current directory. If you specify a directory path, you must make sure that there is a `.sopcinfo` file in the directory.
`--separate-masters`	Does not combine a module's masters that are in the same address space.
`--output-dir[=<`*dirname*`>]`	Allows you to specify multiple header files in *dirname*. The default output directory is '.'
`--single[=<`*filename*`>]`	Allows you to create a single header file, *filename*.
`--single-prefix[=<`*prefix*`>]`	Prefixes macros from a selected single master.
`--module[=<`*moduleName*`>]`	Specifies the module name when creating a single header file.
`--master[=<`*masterName*`>]`	Specifies the master name when creating a single header file.
`--format[=<`*type*`>]`	Specifies the header file format. Default file format is `.h`.
`--silent`	Does not display normal messages.
`--help`	Displays help for `sopc-create-header-files`.

By default, the `sopc-create-header-files` command creates multiple header files. There is one header file for the entire system, and one header file for each master group in each module. A master group is a set of masters in a module in the same address space. In general, a module may have multiple master groups. Addresses and available devices are a function of the master group.

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

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

Table 101. Supported Header File Formats

Type	Suffix	Uses	Example
h	`.h`	C/C++ header files	`#define FOO 12`
m4	`.m4`	Macro files for m4	`m4_define("FOO", 12)`
sh	`.sh`	Shell scripts	`FOO=12`
mk	`.mk`	Makefiles	`FOO := 12`
pm	`.pm`	Perl scripts	`$macros{FOO} = 12;`

Note: You can use the `sopc-create-header-files` command when you want to generate C macro files for DMAs that have access to memory that the Nios II does not have access to.

9.14.9 Incrementally Generate the System

You can modify the parameters of an IP component and regenerate the RTL for just that particular IP component.

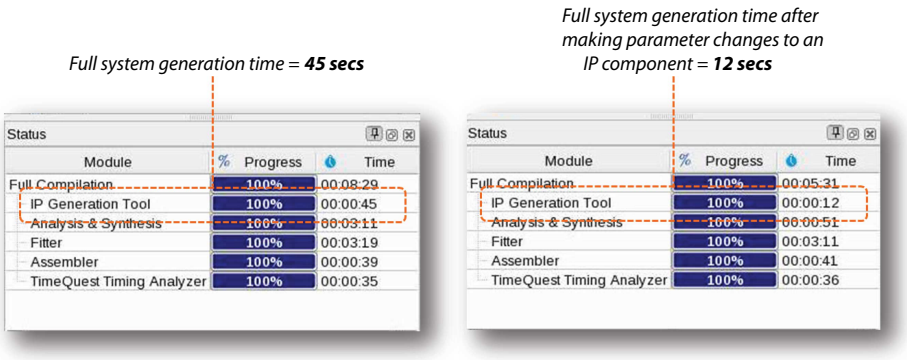
The example below demonstrates the incremental generation flow of a Qsys Pro System:

1. In Qsys Pro, click **File ➤ New System**. The **Create New System** dialog box appears, from which you create your new Qsys Pro system and associate your system with a specific Quartus Prime project.
2. In the IP Catalog search box, locate the **On-Chip Memory (RAM or ROM)** and double-click to add the component to your system.
3. Similarly, locate the **Reset Bridge** and **Clock Bridge** components and double-click to add the components to your system.
4. Make the necessary system connections between the IP components added to the system.

   Note: For more information about connecting IP components, refer to *Connecting IP Components*.
5. To save and close the system without generating, click **File ➤ Save**.
6. In the Quartus Prime software, click **File ➤ Open Project**.
7. Select the Quartus Prime project associated with your saved Qsys Pro system. The Quartus Prime software opens the project and the associated Qsys Pro system.
8. To start the compilation of the Quartus Prime project, click **Processing ➤ Start Compilation**.
9. To open the Status window, click **View ➤ Status**. From this window, track the time for Full Compilation, as well as IP components Generation.
10. Once the compilation finishes, in Qsys Pro, click **File ➤ Open**.
11. Select the `.ip` file for any one of the IP components in your saved system.
12. Modify some parameter in this `.ip` file.

    Note: Make sure your modifications do not affect the parent system, requiring a system update by running **Validate System Integrity** from within the Qsys Pro system after loading the parent system, or by running `qsys-validate` from the command-line.
13. To save the IP file, click **File ➤ Save**.
14. To restart the compilation of the same Quartus Prime project with modified Qsys Pro system, click **Processing ➤ Start Compilation** in the Quartus Prime software. Qsys Pro generates the RTL only for the modified IP component, skipping the generation of the other components in the system.

Figure 158. Incremental Generation of Qsys Pro System



Related Links

Connect IP Components in Your Qsys Pro System on page 311
Use the **System Contents** tab to connect and configure components. Qsys Pro supports connections between interfaces of compatible types and opposite directions. For example, you can connect a memory-mapped master interface to a slave interface, and an interrupt sender interface to an interrupt receiver interface. You can connect any interfaces exported from a Qsys Pro system within a parent system.

9.15 Explore and Manage Qsys Pro Interconnect

The System with Qsys Pro Interconnect window allows you to see the contents of the Qsys Pro interconnect before you generate your system. In this display of your system, you can review a graphical representation of the generated interconnect. Qsys Pro converts connections between interfaces to interconnect logic during system generation.

You access the System with Qsys Pro Interconnect window by clicking **Show System With Qsys Pro Interconnect** command on the **System** menu.

The System with Qsys Pro Interconnect window has the following tabs:

- **System Contents**—Displays the original instances in your system, as well as the inserted interconnect instances. Connections between interfaces are replaced by connections to interconnect where applicable.
- **Hierarchy**—Displays a system hierarchical navigator, expanding the system contents to show modules, interfaces, signals, contents of subsystems, and connections.
- **Parameters**—Displays the parameters for the selected element in the **Hierarchy** tab.
- **Memory-Mapped Interconnect**—Allows you to select a memory-mapped interconnect module and view its internal command and response networks. You can also insert pipeline stages to achieve timing closure.

The **System Contents**, **Hierarchy**, and **Parameters** tabs are read-only. Edits that you apply on the **Memory-Mapped Interconnect** tab are automatically reflected on the **Interconnect Requirements** tab.

The **Memory-Mapped Interconnect** tab in the System with Qsys Pro Interconnect window displays a graphical representation of command and response datapaths in your system. Datapaths allow you precise control over pipelining in the interconnect. Qsys Pro displays separate figures for the command and response datapaths. You can access the datapaths by clicking their respective tabs in the **Memory-Mapped Interconnect** tab.

Each node element in a figure represents either a master or slave that communicates over the interconnect, or an interconnect sub-module. Each edge is an abstraction of connectivity between elements, and its direction represents the flow of the commands or responses.

Click **Highlight Mode** (**Path**, **Successors**, **Predecessors**) to identify edges and datapaths between modules. Turn on **Show Pipelinable Locations** to add greyed-out registers on edges where pipelining is allowed in the interconnect.

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

9.15.1 Manually Controlling Pipelining in the Qsys Pro Interconnect

The **Memory-Mapped Interconnect** tab allows you to manipulate pipleline connections in the Qsys Pro interconnect. Access the **Memory-Mapped Interconnect** tab by clicking **System ➤ Show System With Qsys Pro Interconnect**

Note: To increase interconnect frequency, you should first try increasing the value of the **Limit interconnect pipeline stages to** option on the **Interconnect Requirements** tab. You should only consider manually pipelining the interconnect if changes to this option do not improve frequency, and you have tried all other options to achieve timing closure, including the use of a bridge. Manually pipelining the interconnect should only be applied to complete systems.

1. In the **Interconnect Requirements** tab, first try increasing the value of the **Limit interconnect pipeline stages to** option until it no longer gives significant improvements in frequency, or until it causes unacceptable effects on other parts of the system.
2. In the Quartus Prime software, compile your design and run timing analysis.
3. Using the timing report, identify the critical path through the interconnect and determine the approximate mid-point. The following is an example of a timing report:

```
2.800 0.000 cpu_instruction_master|out_shifter[63]|q
3.004 0.204 mm_domain_0|addr_router_001|Equal5~0|datac
3.246 0.242 mm_domain_0|addr_router_001|Equal5~0|combout
3.346 0.100 mm_domain_0|addr_router_001|Equal5~1|dataa
3.685 0.339 mm_domain_0|addr_router_001|Equal5~1|combout
4.153 0.468 mm_domain_0|addr_router_001|src_channel[5]~0|datad
4.373 0.220 mm_domain_0|addr_router_001|src_channel[5]~0|combout
```

4. In Qsys Pro, click **System ➤ Show System With Qsys Pro Interconnect**.
5. In the **Memory-Mapped Interconnect** tab, select the interconnect module that contains the critical path. You can determine the name of the module from the hierarchical node names in the timing report.

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

Manual pipelining has the following limitations:

- If you make changes to your original system's connectivity after manually pipelining an interconnect, your inserted pipelines may become invalid. Qsys Pro displays warning messages when you generate your system if invalid pipeline stages are detected. You can remove invalid pipeline stages with the **Remove Stale Pipelines** option in the **Memory-Mapped Interconnect** tab. Intel recommends that you do not make changes to the system's connectivity after manual pipeline insertion.
- Review manually-inserted pipelines when upgrading to newer versions of Qsys Pro. Manually-inserted pipelines in one version of Qsys Pro may not be valid in a future version.

Related Links

- Specify Qsys Pro Interconnect Requirements on page 346
  The **Interconnect Requirements** tab allows you to apply system-wide, `$system`, and interface interconnect requirements for IP components in your system.
- Qsys Pro System Design Components on page 879
  You can use Qsys Pro IP components to create Qsys Pro systems.

9.16 Implement Performance Monitoring

Use the Qsys Pro **Instrumentation** tab (**View ➤ Instrumentation**) in to set up real-time performance monitoring using throughput metrics such as read and write transfers. The **Add debug instrumentation to the Qsys Pro Interconnect** option allows you to interact with the Bus Analyzer Toolkit, which you can access on the **Tools** menu in the Quartus Prime software.

Qsys Pro supports performance monitoring for only Avalon-MM interfaces. In your Qsys Pro system, you can monitor the performance of no less than three, and no greater than 15 components at one time. The performance monitoring feature works with Quartus Prime software devices 13.1 and newer.

Note: For more information about the Bus Analyzer Toolkit and the Qsys Pro `Instrumentation` tab, refer to the **Bus Analyzer Toolkit** page.

Related Links

Bus Analyzer Toolkit

9.17 Qsys Pro 64-Bit Addressing Support

Qsys Pro interconnect supports up to 64-bit addressing for all Qsys Pro interfaces and IP components, with a range of: `0x0000 0000 0000 0000` to `0xFFFF FFFF FFFF FFFF`, inclusive.

Address parameters appear in the **Base** and **End** columns in the **System Contents** tab, on the **Address Map** tab, in the parameter editor, and in validation messages. Qsys Pro displays as many digits as needed in order to display the top-most set bit, for example, 12 hex digits for a 48-bit address.

A Qsys Pro system can have multiple 64-bit masters, with each master having its own address space. You can share slaves between masters and masters can map slaves to different addresses. For example, one master can interact with slave `0` at base address `0000_0000_0000`, and another master can see the same slave at base address `c000_000_000`.

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

Related Links

Address Span Extender on page 897
The **Address Span Extender** allows memory-mapped master interfaces to access a larger or smaller address map than the width of their address signals allows.

9.17.1 Support for Avalon-MM Non-Power of Two Data Widths

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

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

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

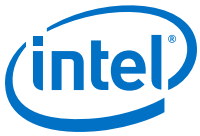
9.18 Qsys Pro System Example Designs

Click the **Example Design** button in the parameter editor to generate an example design.

If there are multiple example designs for an IP component, then there is a button for each example in the parameter editor. When you click the **Example Design** button, the **Select Example Design Directory** dialog box appears, where you can select the directory to save the example design.

The **Example Design** button does not appear in the parameter editor if there is no example. For some IP components, you can click **Generate ➤ Generate Example Design** to access an example design.

The following Qsys Pro system example designs demonstrate various design features and flows that you can replicate in your Qsys Pro system.



Related Links

- Nios II Qsys Pro Example Design
- PCI Express Avalon-ST Qsys Pro Example Design
- Triple Speed Ethernet Qsys Pro Example Design

9.19 Qsys Pro Command-Line Utilities

You can perform many of the functions available in the Qsys Pro GUI at the command-line, with Qsys Pro command-line utilities.

You run Qsys Pro command-line executables from the Quartus Prime installation directory:

```
<Quartus Prime installation directory>\quartus\sopc_builder\bin
```

For command-line help listing of all the options for any executable, type the following command:

```
<Quartus Prime installation directory>\quartus\sopc_builder\bin
\<executable name> --help
```

Note: You must add `$QUARTUS_ROOTDIR/sopc_builder/bin/` to the PATH variable to access command-line utilities. Once you add this PATH variable, you can launch the utility from any directory location.

9.19.1 Run the Qsys Pro Editor with qsys-edit

You can use the `qsys-edit` utility to run the Qsys Pro editor from command-line.

You can use the following options with the `qsys-edit` utility:

Table 102. qsys-edit Command-Line Options

Option	Usage	Description
1st arg file	Optional	Specifies the name of the `.qsys` system or `.qvar` variation file to edit.
`--search-path[=<value>]`	Optional	If you omit this command, Qsys Pro uses a standard default path. If you provide a search path, Qsys Pro searches a comma-separated list of paths. To include the standard path in your replacement, use "$", for example: `/extra/dir,$`.
`--quartus-project[=<value>]`	Required	This option is mandatory if you are associating your Qsys Pro system with an existing Quartus Prime project. Specifies the name of the Quartus Prime project file. If you do not provide the revision via `--rev`, Qsys Pro uses the default revision as the Quartus Prime project name.
`--new-quartus-project[=<value>]`	Required	This option is mandatory if you are associating your Qsys Pro system with a new Quartus Prime project. Specifies the name and path of the new Quartus Prime project. Creates a new Quartus Prime project at the specified path. You can also provide the revision name.
`--rev[=<value>]`	Optional	Specifies the name of the Quartus Prime project revision.

continued...

Option	Usage	Description
`--family[=<value>]`	Optional	Sets the device family.
`--part[=<value>]`	Optional	Sets the device part number. If set, this option overrides the `--family` option.
`--new-component-type[=<value>]`	Optional	Specifies the instance type for parameterization in a variation.
`--require-generation`	Optional	Marks the loading system as requiring generation.
`--debug`	Optional	Enables debugging features and output.
`--jvm-max-heap-size=<value>`	Optional	The maximum memory size that Qsys Pro uses when running `qsys-edit`. You specify this value as `<size><unit>`, where unit is `m` (or `M`) for multiples of megabytes, or `g` (or `G`) for multiples of gigabytes. The default value is `512m`.
`--help`	Optional	Displays help for `qsys-edit`.

9.19.2 Scripting IP Core Generation

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

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

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

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

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

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

Table 103. qsys-generate Command-Line Options

Option	Usage	Description
<1st arg file>	Required	Specifies the name of the `.qsys` system file to generate.
`--synthesis=<VERILOG|VHDL>`	Optional	Creates synthesis HDL files that Qsys Pro uses to compile the system in a Quartus Prime project. Specify the preferred generation language for the top-level RTL file for the generated Qsys Pro system. The default value is *VERILOG*.
`--block-symbol-file`	Optional	Creates a Block Symbol File (**.bsf**) for the Qsys Pro system.
`--greybox`	Optional	If you are synthesizing your design with a third-party EDA synthesis tool, generate a netlist for the synthesis tool to estimate timing and resource usage for this design.
`--ipxact`	Optional	If you set this option to true, Qsys Pro renders the post-generation system as an IPXACT-compatible component description.

continued...

Option	Usage	Description
`--simulation=<VERILOG|VHDL>`	Optional	Creates a simulation model for the Qsys Pro system. The simulation model contains generated HDL files for the simulator, and may include simulation-only features. Specify the preferred simulation language. The default value is *VERILOG*.
`--testbench=<SIMPLE|STANDARD>`	Optional	Creates a testbench system that instantiates the original system, adding bus functional models (BFMs) to drive the top-level interfaces. When you generate the system, the BFMs interact with the system in the simulator. The default value is *STANDARD*.
`--testbench-simulation=<VERILOG|VHDL>`	Optional	After you create the testbench system, create a simulation model for the testbench system. The default value is *VERILOG*.
`--example-design=<value>`	Optional	Creates example design files. For example, `--example-design` or `--example-design=all`. The default is *All*, which generates example designs for all instances. Alternatively, choose specific filesets based on instance name and fileset name. For example `--example-design=instance0.example_design1,instance1.example_design 2`. Specify an output directory for the example design files creation.
`--search-path=<value>`	Optional	If you omit this command, Qsys Pro uses a standard default path. If you provide this command, Qsys Pro searches a comma-separated list of paths. To include the standard path in your replacement, use "$", for example, `"/extra/dir,$"`.
`--family=<value>`	Optional	Sets the device family name.
`--part=<value>`	Optional	Sets the device part number. If set, this option overrides the `--family` option.
`--upgrade-variation-file`	Optional	If you set this option to true, the file argument for this command accepts a `.v` file, which contains a IP variant. This file parameterizes a corresponding instance in a Qsys Pro system of the same name.
`--upgrade-ip-cores`	Optional	Enables upgrading all the IP cores that support upgrade in the Qsys Pro system.
`--clear-output-directory`	Optional	Clears the output directory corresponding to the selected target, that is, simulation or synthesis.
`--jvm-max-heap-size=<value>`	Optional	The maximum memory size that Qsys Pro uses when running `qsys-generate`. You specify the value as `<size><unit>`, where `unit` is m (or M) for multiples of megabytes or g (or G) for multiples of gigabytes. The default value is 512m.
`--help`	Optional	Displays help for `--qsys-generate`.

9.19.3 Display Available IP Components with ip-catalog

The `ip-catalog` command displays a list of available IP components relative to the current Quartus Prime project directory, as either text or XML.

You can use the following options with the `ip-catalog` utility:

Table 104. ip-catalog Command-Line Options

Option	Usage	Description
`--project-dir=<directory>`	Optional	Finds IP components relative to the Quartus Prime project directory. By default, Qsys Pro uses '.' as the current directory. To exclude a project directory, leave the value empty.
`--type`	Optional	Provides a pattern to filter the type of available plug-ins. By default, Qsys Pro shows only IP components. To look for a partial type string, surround with *, for instance, *connection*.
`--name=<value>`	Optional	Provides a pattern to filter the names of the IP components found. To show all IP components, use a * or ' '. By default, Qsys Pro shows all IP components. The argument is not case sensitive. To look for a partial name, surround with *, for instance, *uart*
`--verbose`	Optional	Reports the progress of the command.
`--xml`	Optional	Generates the output in XML format, in place of colon-delimited format.
`--search-path=<value>`	Optional	If you omit this command, Qsys Pro uses a standard default path. If you provide this command, Qsys Pro searches a comma-separated list of paths. To include the standard path in your replacement, use "$", for example, "/extra/dir,$".
`<1st arg value>`	Optional	Specifies the directory or name fragment.
`--jvm-max-heap-size=<value>`	Optional	The maximum memory size that Qsys Pro uses for when running `ip-catalog`. You specify the value as `<size><unit>`, where `unit` is m (or M) for multiples of megabytes or g (or G) for multiples of gigabytes. The default value is 512m.
`--help`	Optional	Displays help for the `ip-catalog` command.

9.19.4 Create an .ipx File with ip-make-ipx

The `ip-make-ipx` command creates an **.ipx** index file. This file provides a convenient way to include a collection of IP components from an arbitrary directory. You can edit the **.ipx** file to disable visibility of one or more IP components in the IP Catalog.

You can use the following options with the `ip-make-ipx` utility:

Table 105. ip-make-ipx Command-Line Options

Option	Usage	Description
`--source-directory=<directory>`	Optional	Specifies the directory containing your IP components. The default directory is '.'. You can provide a comma-separated list of directories.
`--output=<file>`	Optional	Specifies the name of the index file to generate. The default name is `/component.ipx`. Set as `--output=<"">` to print the output to the console.
`--relative-vars=<value>`	Optional	Causes the output file to include references relative to the specified variable(s) wherever possible. You can specify multiple variables as a comma-separated list.
`--thorough-descent`	Optional	If you set this option, Qsys Pro searches all the component files, without skipping the sub-directories.

continued...

Option	Usage	Description
`--message-before=<value>`	Optional	Prints a log message at the start of reading an index file.
`--message-after=<value>`	Optional	Prints a log message at the end of reading an index file.
`--jvm-max-heap-size=<value>`	Optional	The maximum memory size Qsys Pro uses when running `ipr-make-ipx`. You specify this value as `<size><unit>`, where unit is `m` (or `M`) for multiples of megabytes, or `g` (or `G`) for multiples of gigabytes. The default value is `512m`.
`--help`	Optional	Displays help for the `ip-make-ipx` command.

Related Links

Set up the IP Index File (.ipx) to Search for IP Components on page 304
An IP Index File (**.ipx**) contains a search path that Qsys Pro uses to search for IP components. You can use the `ip-make-ipx` command to create an **.ipx** file for any directory tree, which can reduce the startup time for Qsys Pro.

9.19.5 Generate Simulation Scripts

You can use the `ip-make-simscript` utility to generate simulation scripts for one or more simulators, given one or more **Simulation Package Descriptor** file(s).

You can use the following options with the `ip-make-simscript` utility:

Table 106. ip-make-simscript Command-Line Options

Option	Usage	Description
`--spd[=<file>]`	Required/Repeatable	The SPD files describe the list of files that require compilation, and memory models hierarchy. This argument can either be a single path to an SPD file or a comma-separated list of paths of SPD files. For instance, `--spd=ipcore_1.spd,ipcore_2.spd`
`--output-directory[=<directory>]`	Optional	Specifies the directory path for the location of output files. If you do not specify a directory, the output directory defaults to the directory from which `--ip-make-simscript` runs.
`--compile-to-work`	Optional	Compiles all design files to the default library - `work`.
`--use-relative-paths`	Optional	Uses relative paths whenever possible.
`--cache-file[=<file>]`	Optional	Generates cache file for managed flow.
--quiet	Optional	Quiet reporting mode. Does not report generated files.
`--jvm-max-heap-size=<value>`	Optional	The maximum memory size Qsys Pro uses when running `ip-make-simscript`. You specify this value as `<size><unit>`, where unit is `m` (or `M`) for multiples of megabytes, or `g` (or `G`) for multiples of gigabytes. The default value is `512m`.
`--help`	Optional	Displays help for `--ip-make-simscript`.

9.19.6 Generate a Qsys Pro System with qsys-script

You can use the `qsys-script` utility to create and manipulate a Qsys Pro system with Tcl scripting commands. If you specify a system, Qsys Pro loads that system before executing any of the scripting commands.

Note: You must provide a package version for the `qsys-script`. If you do not specify the `--package-version=<value>` command, you must then provide a Tcl script and request the system scripting API directly with the `package require -exact qsys < version >` command.

Example 83. Qsys Pro Command-Line Scripting Example

```
qsys-script --script=my_script.tcl \
--system-file=fancy.qsys my_script.tcl contains:
package require -exact qsys 16.0
# get all instance names in the system and print one by one
set instances [ get_instances ]
foreach instance $instances {
    send_message Info "$instance"
}
```

You can use the following options with the `qsys-script` utility:

Table 107. qsys-script Command-Line Options

Option	Usage	Description
`--system-file=<file>`	Optional	Specifies the path to a `.qsys` file. Qsys Pro loads the system before running scripting commands.
`--script=<file>`	Optional	A file that contains Tcl scripting commands that you can use to create or manipulate a Qsys Pro system. If you specify both `--cmd` and `--script`, Qsys Pro runs the `--cmd` commands before the script specified by `--script`.
`--cmd=<value>`	Optional	A string that contains Tcl scripting commands that you can use to create or manipulate a Qsys Pro system. If you specify both `--cmd` and `--script`, Qsys Pro runs the `--cmd` commands before the script specified by `--script`.
`--package-version=<value>`	Optional	Specifies which Tcl API scripting version to use and determines the functionality and behavior of the Tcl commands. The Quartus Prime software supports Tcl API scripting commands. The minimum supported version is 12.0. If you do not specify the version on the command-line, your script must request the scripting API directly with the `package require -exact qsys` *<version >* command.
`--search-path=<value>`	Optional	If you omit this command, a Qsys Pro uses a standard default path. If you provide this command, Qsys Pro searches a comma-separated list of paths. To include the standard path in your replacement, use "$", for example, `/< directory path >/dir,$`. Separate multiple directory references with a comma.
`--quartus-project=<value>`	Optional	Specifies the path to a `.qpf` Quartus Prime project file. Utilizes the specified Quartus Prime project to add the file saved using `save_system` command. If you omit this command, Qsys Pro uses the default revision as the project name.
`--new-quartus-project=<value>`	Optional	Specifies the name of the new Quartus Prime project. Creates a new Quartus Prime project at the specified path and adds the file saved using `save_system` command to the project. If you omit this command, Qsys Pro uses the Quartus Prime project revision as the new Quartus Prime project name.

continued...

Option	Usage	Description
`--rev=<value>`	Optional	Allows you to specify the name of the Quartus Prime project revision.
`--jvm-max-heap-size=<value>`	Optional	The maximum memory size that the `qsys-script` tool uses. You specify this value as `<size><unit>`, where unit is `m` (or `M`) for multiples of megabytes, or `g` (or `G`) for multiples of gigabytes.
`--help`	Optional	Displays help for the `qsys-script` utility.

Related Links

Altera Wiki Qsys Pro Scripts

9.19.6.1 Qsys Pro Scripting Command Reference

Qsys Pro system scripting provides Tcl commands to manipulate your system. The `qsys-script` provides a command-line alternative to the Qsys Pro tool. Use the `qsys-script` commands to create and modify your system, as well as to create reports about the system.

To use the current version of the Tcl commands, include the following line at the top of your script:

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

For example, for the current release of the Quartus Prime software, include:

```
package require -exact qsys 17.0
```

The Qsys Pro scripting commands fall under the following categories:

System on page 379
This section lists the commands that allow you to manipulate your Qsys Pro system.

Subsystems on page 392
This section lists the commands that allow you to obtain the connection and parameter information of instances in your Qsys Pro subsystem.

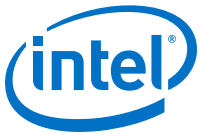
Instances on page 401
This section lists the commands that allow you to manipulate the instances of IP components in your Qsys Pro system.

Instantiations on page 434
This section lists the commands that allow you to manipulate the loaded instantiations in your Qsys Pro system.

Components on page 473
This section lists the commands that allow you to manipulate the loaded IP components in your Qsys Pro system.

Connections on page 499
This section lists the commands that allow you to manipulate the interface connections in your Qsys Pro system.

Top-level Exports on page 511
This section lists the commands that allow you to manipulate the exported interfaces in your Qsys Pro system.



9.19.6.1.1 System

This section lists the commands that allow you to manipulate your Qsys Pro system.

create_system

Description

Replaces the current system with a new system of the specified name.

Usage

`create_system [<name>]`

Returns

No return value.

Arguments

name (optional) The new system name.

Example

```
create_system my_new_system_name
```

Related Links

- load_system on page 388
- save_system on page 389

export_hw_tcl

Description

Allows you to save the currently open system as an `_hw.tcl` file in the project directory. The saved systems appears under the **System** category in the IP Category.

Usage

```
export_hw_tcl
```

Returns

No return value.

Arguments

No arguments

Example

```
export_hw_tcl
```

Related Links

- load_system on page 388
- save_system on page 389

get_device_families

Description

Returns the list of installed device families.

Usage

```
get_device_families
```

Returns

String[] The list of device families.

Arguments

No arguments

Example

```
get_device_families
```

Related Links

get_devices on page 383

get_devices

Description

Returns the list of installed devices for the specified family.

Usage

`get_devices <family>`

Returns

String[] The list of devices.

Arguments

family Specifies the family name to get the devices for.

Example

```
get_devices exampleFamily
```

Related Links

get_device_families on page 382

get_module_properties

Description

Returns the properties that you can manage for a top-level module of the Qsys Pro system.

Usage

```
get_module_properties
```

Returns

The list of property names.

Arguments

No arguments.

Example

```
get_module_properties
```

Related Links

- get_module_property on page 385
- set_module_property on page 390

get_module_property

Description

Returns the value of a top-level system property.

Usage

`get_module_property <property>`

Returns

The property value.

Arguments

property The property name to query. Refer to *Module Properties*.

Example

```
get_module_property NAME
```

Related Links

- get_module_properties on page 384
- set_module_property on page 390

get_project_properties

Description

Returns the list of properties that you can query for properties pertaining to the Quartus Prime project.

Usage

```
get_project_properties
```

Returns

The list of project properties.

Arguments

No arguments

Example

```
get_project_properties
```

Related Links

- get_project_property on page 387
- set_project_property on page 391

get_project_property

Description

Returns the value of a Quartus Prime project property.

Usage

`get_project_property <property>`

Returns

The property value.

Arguments

property The project property name. Refer to *Project properties*.

Example

```
get_project_property DEVICE_FAMILY
```

Related Links

- get_module_properties on page 384
- get_module_property on page 385
- set_module_property on page 390
- Project Properties on page 564

load_system

Description

Loads the Qsys Pro system from a file, and uses the system as the current system for scripting commands.

Usage

`load_system` <*file*>

Returns

No return value.

Arguments

file The path to the `.qsys` file.

Example

```
load_system example.qsys
```

Related Links

- create_system on page 380
- save_system on page 389

save_system

Description

Saves the current system to the specified file. If you do not specify the file, Qsys Pro saves the system to the same file opened with the `load_system` command.

Usage

`save_system` <*file*>

Returns

No return value.

Arguments

file If available, the path of the **.qsys** file to save.

Example

```
save_system
```

```
save_system file.qsys
```

Related Links

- load_system on page 388
- create_system on page 380

set_module_property

Description

Specifes the Tcl procedure to evaluate changes in Qsys Pro system instance parameters.

Usage

`set_module_property` <*property*> <*value*>

Returns

No return value.

Arguments

property The property name. Refer to *Module Properties*.

value The new value of the property.

Example

```
set_module_property COMPOSITION_CALLBACK "my_composition_callback"
```

Related Links

- get_module_properties on page 384
- get_module_property on page 385
- Module Properties on page 558

set_project_property

Description

Sets the project property value, such as the device family.

Usage

`set_project_property` <*property*> <*value*>

Returns

No return value.

Arguments

property The property name. Refer to *Project Properties*.

value The new property value.

Example

```
set_project_property DEVICE_FAMILY "Cyclone IV GX"
```

Related Links

- get_project_properties on page 386
- get_project_property on page 387
- Project Properties on page 564

9.19.6.1.2 Subsystems

This section lists the commands that allow you to obtain the connection and parameter information of instances in your Qsys Pro subsystem.

get_composed_connections

Description

Returns the list of all connections in the subsystem for an instance that contains the subsystem of the Qsys Pro system.

Usage

`get_composed_connections` <*instance*>

Returns

The list of connection names in the subsystem.

Arguments

instance The child instance containing the subsystem.

Example

```
get_composed_connections subsystem_0
```

Related Links

- get_composed_connection_parameter_value on page 394
- get_composed_connection_parameters on page 395

get_composed_connection_parameter_value

Description

Returns the parameter value of a connection in a child instance containing the subsystem.

Usage

`get_composed_connection_parameter_value` <*instance*> <*child_connection*> <*parameter*>

Returns

The parameter value.

Arguments

instance The child instance that contains the subsystem.

child_connection The connection name in the subsystem.

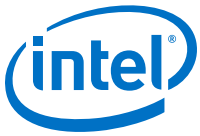
parameter The parameter name to query for the connection.

Example

```
get_composed_connection_parameter_value subsystem_0 cpu.data_master/memory.s0
baseAddress
```

Related Links

- get_composed_connection_parameters on page 395
- get_composed_connections on page 393



get_composed_connection_parameters

Description

Returns the list of parameters of a connection in the subsystem, for an instance that contains the subsystem.

Usage

`get_composed_connection_parameters` <*instance*> <*child_connection*>

Returns

The list of parameter names.

Arguments

instance The child instance containing the subsystem.

child_connection The name of the connection in the subsystem.

Example

```
get_composed_connection_parameters subsystem_0 cpu.data_master/memory.s0
```

Related Links

- get_composed_connection_parameter_value on page 394
- get_composed_connections on page 393

get_composed_instance_assignment

Description

Returns the assignment value of the child instance in the subsystem.

Usage

`get_composed_instance_assignment` <*instance*> <*child_instance*> <*assignment*>

Returns

The assignment value.

Arguments

instance The subsystem containing the child instance.

child_instance The child instance name in the subsystem.

assignment The assignment key.

Example

```
get_composed_instance_assignment subsystem_0 video_0
"embeddedsw.CMacro.colorSpace"
```

Related Links

- get_composed_instance_assignments on page 397
- get_composed_instances on page 400

get_composed_instance_assignments

Description

Returns the list of assignments of the child instance in the subsystem.

Usage

`get_composed_instance_assignments` <*instance*> <*child_instance*>

Returns

The list of assignment names.

Arguments

instance The subsystem containing the child instance.

child_instance The child instance name in the subsystem.

Example

```
get_composed_instance_assignments subsystem_0 cpu
```

Related Links

- get_composed_instance_assignment on page 396
- get_composed_instances on page 400

get_composed_instance_parameter_value

Description

Returns the parameter value of the child instance in the subsystem.

Usage

`get_composed_instance_parameter_value` <*instance*> <*child_instance*> <*parameter*>

Returns

The parameter value of the instance in the subsystem.

Arguments

instance The subsystem containing the child instance.

child_instance The child instance name in the subsystem.

parameter The parameter name to query on the child instance in the subsystem.

Example

```
get_composed_instance_parameter_value subsystem_0 cpu DATA_WIDTH
```

Related Links

- get_composed_instance_parameters on page 399
- get_composed_instances on page 400

get_composed_instance_parameters

Description

Returns the list of parameters of the child instance in the subsystem.

Usage

`get_composed_instance_parameters` <*instance*> <*child_instance*>

Returns

The list of parameter names.

Arguments

instance The subsystem containing the child instance.

child_instance The child instance name in the subsystem.

Example

```
get_composed_instance_parameters subsystem_0 cpu
```

Related Links

- get_composed_instance_parameter_value on page 398
- get_composed_instances on page 400

get_composed_instances

Description

Returns the list of child instances in the subsystem.

Usage

`get_composed_instances <instance>`

Returns

The list of instance names in the subsystem.

Arguments

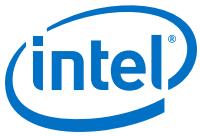
instance The subsystem containing the child instance.

Example

```
get_composed_instances subsystem_0
```

Related Links

- get_composed_instance_assignment on page 396
- get_composed_instance_assignments on page 397
- get_composed_instance_parameter_value on page 398
- get_composed_instance_parameters on page 399



9.19.6.1.3 Instances

This section lists the commands that allow you to manipulate the instances of IP components in your Qsys Pro system.

add_instance

Description

Adds an instance of a component, referred to as a *child* or *child instance*, to the system.

Usage

`add_instance` <*name*> <*type*> [<*version*>]

Returns

No return value.

Arguments

name Specifies a unique local name that you can use to manipulate the instance. Qsys Pro uses this name in the generated HDL to identify the instance.

type Refers to a kind of instance available in the IP Catalog, for example `altera_avalon_uart`.

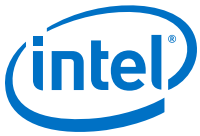
version (optional) The required version of the specified instance type. If you do not specify any instance, Qsys Pro uses the latest version.

Example

```
add_instance uart_0 altera_avalon_uart 16.1
```

Related Links

- get_instance_property on page 427
- get_instances on page 428
- remove_instance on page 430
- set_instance_parameter_value on page 431
- get_instance_parameter_value on page 422



apply_instance_preset

Description

Applies the settings in a preset to the specified instance.

Usage

`apply_instance_preset` <*preset_name*>

Returns

No return value.

Arguments

preset_name The preset name.

Example

```
apply_preset "Custom Debug Settings"
```

Related Links

set_instance_parameter_value on page 431

create_ip

Description

Creates a new IP Variation system with the given instance.

Usage

`create_ip` <*type*> [<*instance_name*> <*version*>]

Returns

No return value.

Arguments

type Kind of instance available in the IP catalog, for example, `altera_avalon_uart`.

instance_name (optional) A unique local name that you can use to manipulate the instance. If not specified, Qsys Pro uses a default name.

version (optional) The required version of the specified instance type. If not specified, Qsys Pro uses the latest version.

Example

```
create_ip altera_avalon_uart altera_avalon_uart_inst 17.0
```

Related Links

- add_component on page 405
- load_system on page 388
- save_system on page 389
- set_instance_parameter_value on page 431

add_component

Description

Adds a new IP Variation component to the system.

Usage

`add_component` <*instance_name*> <*file_name*> [<*component_type*> <*component_instance_name*> <*component_version*>]

Returns

No return value.

Arguments

instance_name A unique local name that you can use to manipulate the instance.

file_name The IP variation file name. If a path is not specified, Qsys Pro saves the file in the `./ip/system/` sub-folder of your system.

component_type (optional) The kind of instance available in the IP catalog, for example `altera_avalon_uart`.

component_instance_name (optional) The instance name of the component in the IP variation file. If not specified, Qsys Pro uses a default name.

component_version (optional) The required version of the specified instance type. If not specified, Qsys Pro uses the latest version.

Example

```
add_component myuart_0 myuart.ip altera_avalon_uart altera_avalon_uart_inst 17.0
```

Related Links

- load_component on page 494
- load_instantiation on page 461
- save_system on page 389

duplicate_instance

Description

Creates a duplicate instance of the specified instance.

Usage

`duplicate_instance` <*instance*> [<*name*>]

Returns

String The new instance name.

Arguments

instance Specifies the instance name to duplicate.

name (*optional*) Specifies the name of the duplicate instance.

Example

```
duplicate_instance cpu cpu_0
```

Related Links

- add_instance on page 402
- remove_instance on page 430

enable_instance_parameter_update_callback

Description

Enables the update callback for instance parameters.

Usage

`enable_instance_parameter_update_callback [<value>]`

Returns

No return value.

Arguments

value (*optional*)	Specifies whether to enable/disable the instance parameters callback. Default option is "1".

Example

```
enabled_instance_parameter_update_callback
```

Related Links

- is_instance_parameter_update_callback_enabled on page 429
- set_instance_parameter_value on page 431

get_instance_assignment

Description

Returns the assignment value of a child instance. Qsys Pro uses assignments to transfer information about hardware to embedded software tools and applications.

Usage

`get_instance_assignment` <*instance*> <*assignment*>

Returns

String The value of the specified assignment.

Arguments

instance The instance name.

assignment The assignment key to query.

Example

```
get_instance_assignment video_0 embeddedsw.CMacro.colorSpace
```

Related Links

get_instance_assignments on page 409

get_instance_assignments

Description

Returns the list of assignment keys for any defined assignments for the instance.

Usage

`get_instance_assignments <instance>`

Returns

String[] The list of assignment keys.

Arguments

instance The instance name.

Example

```
get_instance_assignments sdram
```

Related Links

get_instance_assignment on page 408

get_instance_documentation_links

Description

Returns the list of all documentation links provided by an instance.

Usage

`get_instance_documentation_links` <*instance*>

Returns

String[] The list of documentation links.

Arguments

instance The instance name.

Example

```
get_instance_documentation_links cpu_0
```

Notes

The list of documentation links includes titles and URLs for the links. For instance, a component with a single data sheet link may return:

```
{Data Sheet} {http://url/to/data/sheet}
```

get_instance_interface_assignment

Description

Returns the assignment value for an interface of a child instance. Qsys Pro uses assignments to transfer information about hardware to embedded software tools and applications.

Usage

`get_instance_interface_assignment` <*instance*> <*interface*> <*assignment*>

Returns

String The value of the specified assignment.

Arguments

instance The child instance name.

interface The interface name.

assignment The assignment key to query.

Example

```
get_instance_interface_assignment sdram s1 embeddedsw.configuration.isFlash
```

Related Links

get_instance_interface_assignments on page 412

get_instance_interface_assignments

Description

Returns the list of assignment keys for any assignments defined for an interface of a child instance.

Usage

`get_instance_interface_assignments` <*instance*> <*interface*>

Returns

String[] The list of assignment keys.

Arguments

instance The child instance name.

interface The interface name.

Example

```
get_instance_interface_assignments sdram s1
```

Related Links

get_instance_interface_assignment on page 411

get_instance_interface_parameter_property

Description

Returns the property value for a parameter in an interface of an instance. Parameter properties are metadata about how Qsys Pro uses the parameter.

Usage

`get_instance_interface_parameter_property` <*instance*> <*interface*> <*parameter*> <*property*>

Returns

various The parameter property value.

Arguments

instance The child instance name.

interface The interface name.

parameter The parameter name for the interface.

property The property name for the parameter. Refer to *Parameter Properties*.

Example

```
get_instance_interface_parameter_property uart_0 s0 setupTime ENABLED
```

Related Links

- get_instance_interface_parameters on page 415
- get_instance_interfaces on page 420
- get_parameter_properties on page 542
- Parameter Properties on page 559

get_instance_interface_parameter_value

Description

Returns the parameter value of an interface in an instance.

Usage

`get_instance_interface_parameter_value` <*instance*> <*interface*> <*parameter*>

Returns

various The parameter value.

Arguments

instance The child instance name.

interface The interface name.

parameter The parameter name for the interface.

Example

```
get_instance_interface_parameter_value uart_0 s0 setupTime
```

Related Links

- get_instance_interface_parameters on page 415
- get_instance_interfaces on page 420

get_instance_interface_parameters

Description

Returns the list of parameters for an interface in an instance.

Usage

`get_instance_interface_parameters` <*instance*> <*interface*>

Returns

String[] The list of parameter names for parameters in the interface.

Arguments

instance The child instance name.

interface The interface name.

Example

```
get_instance_interface_parameters uart_0 s0
```

Related Links

- get_instance_interface_parameter_value on page 414
- get_instance_interfaces on page 420

get_instance_interface_port_property

Description

Returns the property value of a port in the interface of a child instance.

Usage

`get_instance_interface_port_property` <*instance*> <*interface*> <*port*> <*property*>

Returns

various The port property value.

Arguments

instance The child instance name.

interface The interface name.

port The port name.

property The property name of the port. Refer to *Port Properties*.

Example

```
get_instance_interface_port_property uart_0 exports tx WIDTH
```

Related Links

- get_instance_interface_ports on page 417
- get_port_properties on page 520
- Port Properties on page 563

get_instance_interface_ports

Description

Returns the list of ports in an interface of an instance.

Usage

`get_instance_interface_ports` <*instance*> <*interface*>

Returns

String[] The list of port names in the interface.

Arguments

instance The instance name.

interface The interface name.

Example

```
get_instance_interface_ports uart_0 s0
```

Related Links

- get_instance_interface_port_property on page 416
- get_instance_interfaces on page 420

get_instance_interface_properties

Description

Returns the list of properties that you can query for an interface in an instance.

Usage

```
get_instance_interface_properties
```

Returns

String[] The list of property names.

Arguments

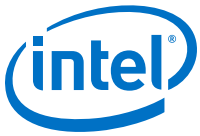
No arguments.

Example

```
get_instance_interface_properties
```

Related Links

- get_instance_interface_property on page 419
- get_instance_interfaces on page 420



get_instance_interface_property

Description

Returns the property value for an interface in a child instance.

Usage

`get_instance_interface_property` <*instance*> <*interface*> <*property*>

Returns

String The property value.

Arguments

instance The child instance name.

interface The interface name.

property The property name. Refer to *Element Properties*.

Example

```
get_instance_interface_property uart_0 s0 DESCRIPTION
```

Related Links

- get_instance_interface_properties on page 418
- get_instance_interfaces on page 420
- Element Properties on page 554

get_instance_interfaces

Description

Returns the list of interfaces in an instance.

Usage

`get_instance_interfaces <instance>`

Returns

String[] The list of interface names.

Arguments

instance The instance name.

Example

```
get_instance_interfaces uart_0
```

Related Links

- get_instance_interface_ports on page 417
- get_instance_interface_properties on page 418
- get_instance_interface_property on page 419

get_instance_parameter_property

Description

Returns the property value of a parameter in an instance. Parameter properties are metadata about how Qsys Pro uses the parameter.

Usage

`get_instance_parameter_property` <*instance*> <*parameter*> <*property*>

Returns

various The parameter property value.

Arguments

instance The instance name.

parameter The parameter name.

property The property name of the parameter. Refer to *Parameter Properties*.

Example

```
get_instance_parameter_property uart_0 baudRate ENABLED
```

Related Links

- get_instance_parameters on page 424
- get_parameter_properties on page 542
- Parameter Properties on page 559

get_instance_parameter_value

Description

Returns the parameter value in a child instance.

Usage

`get_instance_parameter_value` <*instance*> <*parameter*>

Returns

various The parameter value.

Arguments

instance The instance name.

parameter The parameter name.

Example

```
get_instance_parameter_value pixel_converter input_DPI
```

Related Links

- get_instance_parameters on page 424
- set_instance_parameter_value on page 431

get_instance_parameter_values

Description

Returns a list of the parameters' values in a child instance.

Usage

`get_instance_parameter_values` <*instance*> <*parameters*>

Returns

String[] A list of the parameters' value.

Arguments

instance The child instance name.

parameter A list of parameter names in the instance.

Example

```
get_instance_parameter_value uart_0 [list param1 param2]
```

Related Links

- get_instance_parameters on page 424
- set_instance_parameter_value on page 431
- set_instance_parameter_values on page 432

get_instance_parameters

Description

Returns the names of all parameters for a child instance that the parent can manipulate. This command omits derived parameters and parameters that have the SYSTEM_INFO parameter property set.

Usage

`get_instance_parameters` <*instance*>

Returns

instance The list of parameters in the instance.

Arguments

instance The instance name.

Example

```
get_instance_parameters uart_0
```

Related Links

- get_instance_parameter_property on page 421
- get_instance_parameter_value on page 422
- set_instance_parameter_value on page 431

get_instance_port_property

Description

Returns the property value of a port contained by an interface in a child instance.

Usage

`get_instance_port_property` <*instance*> <*port*> <*property*>

Returns

various The property value for the port.

Arguments

instance The child instance name.

port The port name.

property The property name. Refer to *Port Properties*.

Example

```
get_instance_port_property uart_0 tx WIDTH
```

Related Links

- get_instance_interface_ports on page 417
- get_port_properties on page 520
- Port Properties on page 563

get_instance_properties

Description

Returns the list of properties for a child instance.

Usage

```
get_instance_properties
```

Returns

String[] The list of property names for the child instance.

Arguments

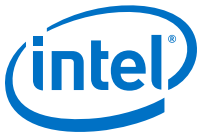
No arguments.

Example

```
get_instance_properties
```

Related Links

get_instance_property on page 427



get_instance_property

Description

Returns the property value for a child instance.

Usage

`get_instance_property` <*instance*> <*property*>

Returns

String The property value.

Arguments

instance The child instance name.

property The property name. Refer to *Element Properties*.

Example

```
get_instance_property uart_0 ENABLED
```

Related Links

- get_instance_properties on page 426
- Element Properties on page 554

get_instances

Description

Returns the list of the instance names for all the instances in the system.

Usage

```
get_instances
```

Returns

String[] The list of child instance names.

Arguments

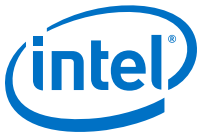
No arguments.

Example

```
get_instances
```

Related Links

- add_instance on page 402
- remove_instance on page 430



is_instance_parameter_update_callback_enabled

Description

Returns true if you enable the update callback for instance parameters.

Usage

```
is_instance_parameter_update_callback_enabled
```

Returns

boolean 1 if you enable the callback; 0 if you disable the callback.

Arguments

No arguments

Example

```
is_instance_parameter_update_callback_enabled
```

Related Links

enable_instance_parameter_update_callback on page 407

remove_instance

Description

Removes an instance from the system.

Usage

`remove_instance <instance>`

Returns

No return value.

Arguments

instance The child instance name to remove.

Example

```
remove_instance cpu
```

Related Links

- add_instance on page 402
- get_instances on page 428

set_instance_parameter_value

Description

Sets the parameter value for a child instance. You cannot set derived parameters and SYSTEM_INFO parameters for the child instance with this command.

Usage

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

Returns

No return value.

Arguments

instance The child instance name.

parameter The parameter name.

value The parameter value.

Example

```
set_instance_parameter_value uart_0 baudRate 9600
```

Related Links

- get_instance_parameter_value on page 422
- get_instance_parameter_property on page 421

set_instance_parameter_values

Description

Sets a list of parameter values for a child instance. You cannot set derived parameters and SYSTEM_INFO parameters for the child instance with this command.

Usage

`set_instance_parameter_value <instance> <parameter_value_pairs>`

Returns

No return value.

Arguments

instance The child instance name.

parameter_value_pairs The pairs of parameter name and value to set.

Example

```
set_instance_parameter_value uart_0 [list baudRate 9600 parity odd]
```

Related Links

- get_instance_parameter_value on page 422
- get_instance_parameter_values on page 423
- get_instance_parameters on page 424

set_instance_property

Description

Sets the property value of a child instance. Most instance properties are read-only and can only be set by the instance itself. The primary use for this command is to update the ENABLED parameter, which includes or excludes a child instance when generating Qsys Pro interconnect.

Usage

`set_instance_property` <*instance*> <*property*> <*value*>

Returns

No return value.

Arguments

instance The child instance name.

property The property name. Refer to *Instance Properties*.

value The property value.

Example

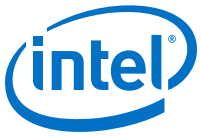
```
set_instance_property cpu ENABLED false
```

Related Links

- get_instance_parameters on page 424
- get_instance_property on page 427
- Instance Properties on page 555

9.19.6.1.4 Instantiations

This section lists the commands that allow you to manipulate the loaded instantiations in your Qsys Pro system.



add_instantiation_hdl_file

Description

Adds an HDL file to the loaded instantiation.

Usage

`add_instantiation_hdl_file <file> [<kind>]`

Returns

No return value.

Arguments

file Specifies the HDL file name.

kind(optional) Indicates the file set kind to add the file to. If you do not specify this option, the command adds the file to all the file sets. Refer to *File Set Kind*.

Example

```
add_instantiation_hdl_file my_nios2_gen2.vhdl quartus_synth
```

Related Links

- load_instantiation on page 461
- save_instantiation on page 465
- File Set Kind on page 570

add_instantiation_interface

Description

Adds an interface to the loaded instantiation.

Usage

`add_instantiation_interface` <*interface*> <*type*> <*direction*>

Returns

No return value.

Arguments

interface Specifies the interface name.

type Specifies the interface type.

direction Specifies the interface direction. Refer to *Interface Direction*.

Example

```
add_instantiation_interface clk_0 clock OUTPUT
```

Related Links

- load_instantiation on page 461
- save_instantiation on page 465
- Interface Direction on page 569

add_instantiation_interface_port

Description

Adds a port to a loaded instantiation's interface.

Usage

`add_instantiation_interface_port` <*interface*> <*port*> <*role*> <*width*> <*vhdl_type*><*direction*>

Returns

No return value.

Arguments

interface Specifies the interface name.

port Specifies the port name.

role Specifies the port role.

width Specifies the port width.

vhdl_type Specifies the VHDL type of the port. Refer to *VHDL Type*.

direction Specifies the port direction. Refer to *Direction Properties*.

Example

```
add_instantiation_interface_port avs_s0 avs_s0_address address 8 {standard logic
vector} input
```

Related Links

- load_instantiation on page 461
- save_instantiation on page 465
- VHDL Type on page 577
- Direction Properties on page 553

copy_instance_interface_to_instantiation

Description

Adds an interface to a loaded instantiation by copying the specified interface of another instance.

Usage

`copy_instance_interface_to_instantiation` <*instance*> <*interface*> <*type*>

Returns

String The name of the newly added interface.

Arguments

instance Specifies the name of the instance to copy the interface from.

interface Specifies the name of the interface to copy.

type Specifies the type of copy to make. Refer to *Instantiation Interface Duplicate Type*.

Example

```
copy_instance_interface_to_instantiation cpu_0 data_master CLONE
```

Related Links

- load_instantiation on page 461
- save_instantiation on page 465
- Instantiation Interface Duplicate Type on page 573

get_instantiation_assignment_value

Description

Gets the assignment value on the loaded instantiation.

Usage

`get_instantiation_assignment_value <name>`

Returns

String The assignment value.

Arguments

name Specifies the name of the assignment to get the value of.

Example

```
get_instantiation_assignment_value embeddedsw.configuration.exceptionOffset
```

Related Links

- load_instantiation on page 461
- save_instantiation on page 465

get_instantiation_assignments

Description

Gets the assignment names in the loaded instantiation.

Usage

```
get_instantiation_assignments
```

Returns

String[] The list of assignment names.

Arguments

No arguments

Example

```
get_instantiation_assignments
```

Related Links

- load_instantiation on page 461
- save_instantiation on page 465

get_instantiation_hdl_file_properties

Description

Returns the list of properties in an HDL file associated with an instantiation.

Usage

```
get_instantiation_hdl_file_properties
```

Returns

String[] The list of property names.

Arguments

No arguments

Example

```
get_instantiation_hdl_file_properties
```

Related Links

- load_instantiation on page 461
- save_instantiation on page 465

get_instantiation_hdl_file_property

Description

Returns the property value of an HDL file associated with the loaded instantiation.

Usage

`get_instantiation_hdl_file_property` <*file*> <*property*>

Returns

various The property value.

Arguments

file Specifies the HDL file name.

property Specifies the property name. Refer to *Instantiation Hdl File Properties*.

Example

```
get_instantiation_hdl_file_property my_nios2_gen2.vhdl OUTPUT_PATH
```

Related Links

- load_instantiation on page 461
- save_instantiation on page 465
- Instantiation Hdl File Properties on page 572

get_instantiation_hdl_files

Description

Returns the list of HDL files of the loaded instantiation.

Usage

`get_instantiation_hdl_files [<kind>]`

Returns

String[] The list of HDL file names.

Arguments

kind (*optional*) Specifies the file set kind to get the files of. If you do not specify this option, the command gets the QUARTUS_SYNTH files. Refer to *File Set Kind*.

Example

```
get_instantiation_hdl_files quartus_synth
```

Related Links

- load_instantiation on page 461
- save_instantiation on page 465
- File Set Kind on page 570

get_instantiation_interface_assignment_value

Description

Gets the assignment value of the loaded instantiation's interface.

Usage

`get_instantiation_interface_assignment_value <interface> <name>`

Returns

String The assignment value

Arguments

interface Specifies the interface name.

name Specifies the assignment name to get the value of.

Example

```
get_instantiation_interface_assignment_value avs_s0
embeddedsw.configuration.exceptionOffset
```

Related Links

- load_instantiation on page 461
- save_instantiation on page 465

get_instantiation_interface_assignments

Description

Gets the assignment names of the loaded instantiation's interface.

Usage

`get_instantiation_interface_assignments` <*interface*>

Returns

String[] The list of assignment names.

Arguments

interface Specifies the interface name.

Example

```
get_instantiation_interface_assignments avs_s0
```

Related Links

- load_instantiation on page 461
- save_instantiation on page 465

get_instantiation_interface_parameter_value

Description

Returns the parameter value of a loaded instantiation's interface.

Usage

`get_instantiation_interface_parameter_value` <*interface*> <*parameter*>

Returns

String The parameter value.

Arguments

interface Specifies the interface name.

parameter Specifies the parameter name.

Example

```
get_instantiation_interface_parameter_value avs_s0 associatedClock
```

Related Links

- get_instantiation_interface_parameters on page 448
- set_instantiation_interface_parameter_value on page 469
- load_instantiation on page 461
- save_instantiation on page 465

get_instantiation_interface_parameters

Description

Returns the list of parameters of an instantiation's interface.

Usage

`get_instantiation_interface_parameters` <*interface*>

Returns

String[] The list of parameter names.

Arguments

interface Specifies the interface name.

Example

```
get_instantiation_interface_parameters avs_s0
```

Related Links

- load_instantiation on page 461
- save_instantiation on page 465
- get_instantiation_interface_parameter_value on page 447
- set_instantiation_interface_parameter_value on page 469

get_instantiation_interface_port_properties

Description

Returns the list of port properties of an instantiation's interface.

Usage

```
get_instantiation_interface_port_properties
```

Returns

String[] The list of port properties.

Arguments

No arguments

Example

```
get_instantiation_interface_port_properties
```

Related Links

- load_instantiation on page 461
- save_instantiation on page 465

get_instantiation_interface_port_property

Description

Returns the port property value of a loaded instantiation's interface.

Usage

`get_instantiation_interface_port_property` <*interface*> <*port*> <*property*>

Returns

various The property value.

Arguments

interface Specifies the interface name.

port Specifies the port name.

property Specifies the property name. Refer to *Port Properties*.

Example

```
get_instantiation_interface_port_property avs_s0 avs_s0_address WIDTH
```

Related Links

- load_instantiation on page 461
- save_instantiation on page 465
- Port Properties on page 576

get_instantiation_interface_ports

Description

Returns the list of ports of the loaded instantiation's interface.

Usage

`get_instantiation_interface_ports` <*interface*>

Returns

String[] The list of port names.

Arguments

interface Specifies the interface name.

Example

```
get_instantiation_interface_ports avs_s0
```

Related Links

- load_instantiation on page 461
- save_instantiation on page 465

get_instantiation_interface_property

Description

Returns the value of a single interface property from the specified instantiation interface.

Usage

`get_instantiation_interface_property` <*interface*> <*property*>

Returns

various The property value.

Arguments

interface The interface name on the currently loaded interface.

property The property name. Refer to *Instantiation Interface Properties*.

Example

```
get_instantiation_interface_property in_clk TYPE
```

Related Links

- get_instantiation_interface_properties on page 453
- load_instantiation on page 461
- Instantiation Interface Properties on page 574

get_instantiation_interface_properties

Description

Returns the names of all the available instantiation interface properties, common to all interface types.

Usage

```
get_instantiation_interface_properties
```

Returns

String[] A list of instantiation interface properties.

Arguments

No arguments.

Example

```
get_instantiation_interface_properties
```

Related Links

get_instantiation_interface_property on page 452

get_instantiation_interface_sysinfo_parameter_value

Description

Gets the system info parameter value for a loaded instantiation's interface.

Usage

`get_instantiation_interface_sysinfo_parameter_value` <*interface*> <*parameter*>

Returns

various The system info property value.

Arguments

interface Specifies the interface name.

parameter Specifies the system info parameter name. Refer to *System Info Type*.

Example

```
get_instantiation_interface_sysinfo_parameter_value debug_mem_slave
max_slave_data_width
```

Related Links

- get_instantiation_interface_sysinfo_parameters on page 455
- set_instantiation_interface_sysinfo_parameter_value on page 471
- System Info Type Properties on page 565

get_instantiation_interface_sysinfo_parameters

Description

Returns the list of system info parameters for the loaded instantiation's interface.

Usage

`get_instantiation_interface_sysinfo_parameters` <*interface*> [<*type*>]

Returns

String[] The list of system info parameter names.

Arguments

interface Specifies the interface name.

type (*optional*) Specifies the parameters type to return. If you do not specify this option, the command returns all the parameters. Refer to *Access Type*.

Example

```
get_instantiation_interface_sysinfo_parameters debug_mem_slave
```

Related Links

- get_instantiation_interface_sysinfo_parameter_value on page 454
- set_instantiation_interface_sysinfo_parameter_value on page 471
- Access Type on page 571

get_instantiation_interfaces

Description

Returns the list of interfaces for the loaded instantiation.

Usage

```
get_instantiation_interfaces
```

Returns

String[] The list of interface names.

Arguments

No arguments.

Example

```
get_instantiation_interfaces
```

Related Links

- load_instantiation on page 461
- save_instantiation on page 465

get_instantiation_properties

Description

Returns the list of properties for the loaded instantiation.

Usage

```
get_instantiation_properties
```

Returns

String[] The list of property names.

Arguments

No arguments.

Example

```
get_instantiation_properties
```

Related Links

- load_instantiation on page 461
- save_instantiation on page 465

get_instantiation_property

Description

Returns the value of the specified property for the loaded instantiation.

Usage

`get_instantiation_property` <*property*>

Returns

various The value of an instantiation property.

Arguments

property Specifies the property name to get the value of. Refer to *Instantiation Properties*.

Example

```
get_instantiation_property HDL_ENTITY_NAME
```

Related Links

- load_instantiation on page 461
- save_instantiation on page 465
- Instantiation Properties on page 575

get_loaded_instantiation

Description

Returns the instance name of the loaded instantiation.

Usage

```
get_loaded_instantiation
```

Returns

String The instance name.

Arguments

No arguments

Example

```
get_loaded_instantiation
```

Related Links

- load_instantiation on page 461
- save_instantiation on page 465

import_instantiation_interfaces

Description

Sets the interfaces of a loaded instantiation, by importing the interfaces from the specified file.

Usage

`import_instantiation_interfaces` <*file*>

Returns

No return value

Arguments

file Specifies the The IP or IP-XACT file to import the interfaces from.

Example

```
import_instantiation_interfaces ip/my_system/my_system_nios2_gen2_0.ip
```

Related Links

- load_instantiation on page 461
- save_instantiation on page 465

load_instantiation

Description

Loads the instantiation of an instance, so that you can modify the instantiation if necessary.

Usage

`load_instantiation` <*instance*>

Returns

boolean 1 if successful; 0 if unsuccessful.

Arguments

instance Specifies the instance name.

Example

```
load_instantiation cpu
```

Related Links

save_instantiation on page 465

remove_instantiation_hdl_file

Description

Removes an HDL file from the loaded instantiation.

Usage

`remove_instantiation_hdl_file <file> [<kind>]`

Returns

No return value.

Arguments

file Specifies the HDL file name.

kind (*optional*) Specifies the kind of file set to remove the file from. If you do not specify this option, the command removes the file from all the file sets. Refer to *File Set Kind*.

Example

```
remove_instantiation_hdl_file my_nios2_gen2.vhdl quartus_synth
```

Related Links

- load_instantiation on page 461
- save_instantiation on page 465
- File Set Kind on page 570

remove_instantiation_interface

Description

Removes an interface from a loaded instantiation.

Usage

`remove_instantiation_interface <interface>`

Returns

No return value

Arguments

interface Specifies the interface name.

Example

```
remove_instantiation_interface avs_s0
```

Related Links

- load_instantiation on page 461
- save_instantiation on page 465

remove_instantiation_interface_port

Description

Removes a port from a loaded instantiation's interface.

Usage

`remove_instantiation_interface_port` <*interface*> <*port*>

Returns

No return value

Arguments

interface Specifies the interface name.

port Specifies the port name.

Example

```
remove_instantiation_interface_port avs_s0 avs_s0_address
```

Related Links

- load_instantiation on page 461
- save_instantiation on page 465

save_instantiation

Description

Saves the loaded instantiation.

Usage

```
save_instantiation
```

Returns

No return value

Arguments

No arguments

Example

```
save_instantiation
```

Related Links

load_instantiation on page 461

set_instantiation_assignment_value

Description

Sets the assignment value for the loaded instantiation.

Usage

`set_instantiation_assignment_value <name> [<value>]`

Returns

No return value

Arguments

instance Specifies the assignment name to set value for.

value (*optional*) Specifies the assignment value. If you do not specify this option, the command removes the assignment.

Example

```
set_instantiation_assignment_value embeddedsw.configuration.exceptionOffset 32
```

Related Links

get_instantiation_assignment_value on page 440

set_instantiation_hdl_file_property

Description

Sets the property value for an HDL file associated with a loaded instantiation.

Usage

`set_instantiation_hdl_file_property<file> <property> <value>`

Returns

No return value

Arguments

file Specifies the HDL file name.

property Specifies the property name. Refer to *Instantiation Hdl File Properties*.

value Specifies the property value.

Example

```
set_instantiation_hdl_file_property my_nios2_gen2.vhdl OUTPUT_PATH
my_nios2_gen2.vhdl
```

Related Links

- load_instantiation on page 461
- save_instantiation on page 465
- Instantiation Hdl File Properties on page 572

set_instantiation_interface_assignment_value

Description

Sets the assignment value for the loaded instantiation's interface.

Usage

`set_instantiation_interface_assignment_value` <*interface*> <*name*> [<*value*>]

Returns

No return value

Arguments

interface Specifies the interface name.

name Specifies the assignment name to set the value of.

value (*optional*) Specifies the new assignment value. If you do not specify this value, the command removes the assignment.

Example

```
set_instantiation_interface_assignment_value
embeddedsw.configuration.exceptionOffset 32
```

Related Links

get_instantiation_assignment_value on page 440

set_instantiation_interface_parameter_value

Description

Sets the parameter value for the loaded instantiation's interface.

Usage

`set_instantiation_interface_parameter_value` <*interface*> <*parameter*> <*value*>

Returns

No return value

Arguments

instance Specifies the interface name.

parameter Specifies the parameter name.

value Specifies the parameter value.

Example

```
set_instantiation_interface_parameter avs_s0 associatedClock clk
```

Related Links

- load_instantiation on page 461
- save_instantiation on page 465
- get_instantiation_interface_parameter_value on page 447
- get_instantiation_interface_parameters on page 448

set_instantiation_interface_port_property

Description

Sets the port property value on a loaded instantiation's interface.

Usage

`set_instantiation_interface_port_property` <*interface*> <*port*> <*property*> <*value*>

Returns

No return value

Arguments

interface Specifies the interface name.

port Specifies the port name.

property Specifies the property name. Refer to *Port Properties*.

value Specifies the property value.

Example

```
set_instantiation_interface_port_property avs_s0 avs_s0_address WIDTH 1
```

Related Links

- load_instantiation on page 461
- save_instantiation on page 465
- Port Properties on page 576

set_instantiation_interface_sysinfo_parameter_value

Description

Sets the system info parameter value for the loaded instantiation's interface.

Usage

`set_instantiation_interface_sysinfo_parameter_value` <*interface*> <*parameter*> <*value*>

Returns

No return value

Arguments

interface Specifies the interface name.

parameter Specifies the system info parameter name. Refer to *System Info Type*.

value Specifies the system info parameter value.

Example

```
set_instantiation_interface_sysinfo_parameter_value debug_mem_slave
max_slave_data_width 64
```

Related Links

- get_instantiation_interface_sysinfo_parameter_value on page 454
- get_instantiation_interface_sysinfo_parameters on page 455
- System Info Type Properties on page 565

set_instantiation_property

Description

Sets the property value for the loaded instantiation.

Usage

`set_instantiation_property` <*property*> <*value*>

Returns

No return value

Arguments

property Specifies the property name. Refer to *Instantiation Properties*.

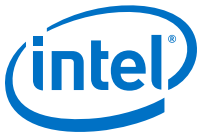
value Specifies the value to set.

Example

```
set_instantiation_property HDL_ENTITY_NAME my_system_nios2_gen2_0
```

Related Links

- load_instantiation on page 461
- save_instantiation on page 465
- Instantiation Properties on page 575



9.19.6.1.5 Components

This section lists the commands that allow you to manipulate the loaded IP components in your Qsys Pro system.

apply_component_preset

Description

Applies the settings in a preset to the loaded component.

Usage

`apply_component_preset<preset_name>`

Returns

No return value

Arguments

preset_name Specifies the preset name.

Example

```
apply_component_preset "Custom Debug Settings"
```

Related Links

- load_component on page 494
- set_component_parameter_value on page 497

get_component_assignment

Description

Returns the assignment value for the loaded component.

Usage

`get_component_assignment <assignment>`

Returns

String The specified assignment value.

Arguments

assignment Specifies the assignment key value to query.

Example

```
get_component_assignment embeddedsw.CMacro.colorSpace
```

Related Links

- load_component on page 494
- get_component_assignments on page 476

get_component_assignments

Description

Returns the list of assignment keys for the loaded component.

Usage

```
get_component_assignments
```

Returns

String[] The list of assignment keys.

Arguments

No arguments

Example

```
get_component_assignments
```

Related Links

- get_instance_assignment on page 408
- load_component on page 494

get_component_documentation_links

Description

Returns the list of all documentation links that the loaded component provides.

Usage

```
get_component_documentation_links
```

Returns

String[] The list of documentation links.

Arguments

No arguments

Example

```
get_component_documentation_links
```

Related Links

load_component on page 494

get_component_interface_assignment

Description

Returns the assignment value of an interface of the loaded component.

Usage

`get_component_interface_assignment` <*interface*> <*assignment*>

Returns

String The specified assignment value.

Arguments

interface Specifies the interface name.

assignment Specifies the assignment key to the query.

Example

```
get_component_interface_assignment s1 embeddedsw.configuration.isFlash
```

Related Links

- get_component_interface_assignments on page 479
- load_component on page 494

get_component_interface_assignments

Description

Returns the list of assignment keys for any assignments that you define for an interface on the loaded component.

Usage

`get_component_interface_assignments <interface>`

Returns

String[] The list of assignment keys.

Arguments

interface Specifies the interface name.

Example

```
get_component_interface_assignments s1
```

Related Links

- get_component_interface_assignment on page 478
- load_component on page 494

get_component_interface_parameter_property

Description

Returns the property value of a parameter in a loaded component's interface. Parameter properties are metadata about how the Quartus Prime uses the parameters.

Usage

`get_component_interface_parameter_property` <*interface*> <*parameter*> <*property*>

Returns

various The parameter property value.

Arguments

interface Specifies the interface name.

parameter Specifies the parameter name.

property Specifies the parameter property. Refer to *Parameter Properties*.

Example

```
get_component_interface_parameter_property s0 setupTime ENABLED
```

Related Links

- get_component_interface_parameters on page 482
- get_component_interfaces on page 486
- load_component on page 494
- Parameter Properties on page 559
- get_parameter_properties on page 542

get_component_interface_parameter_value

Description

Returns the parameter value of an interface of the loaded component.

Usage

`get_component_interface_parameter_value` <*interface*> <*parameter*>

Returns

various The parameter value.

Arguments

interface Specifies the interface name.

parameter Specifies the parameter name.

Example

```
get_component_interface_parameter_value s0 setupTime
```

Related Links

- get_component_interfaces on page 482
- get_component_interfaces on page 486
- load_component on page 494

get_component_interface_parameters

Description

Returns the list of parameters for an interface of the loaded component.

Usage

`get_component_interface_parameters` <*interface*>

Returns

String[] The list of parameter names.

Arguments

interface Specifies the interface name.

Example

```
get_component_interface_parameters s0
```

Related Links

- get_component_interface_parameter_value on page 481
- get_component_interfaces on page 486
- load_component on page 494

get_component_interface_port_property

Description

Returns the property value of a port in the interface of the loaded component.

Usage

`get_component_interface_port_property` <*interface*> <*port*> <*property*>

Returns

various The port property value

Arguments

interface Specifies the interface name.

port Specifies the port name of the interface.

property Specifies the property name of the port. Refer to *Port Properties*.

Example

```
get_component_interface_port_property exports tx WIDTH
```

Related Links

- get_component_interface_ports on page 484
- load_component on page 494
- Port Properties on page 576
- get_port_properties on page 520

get_component_interface_ports

Description

Returns the list of interface ports of the loaded component.

Usage

`get_component_interface_ports` <*interface*>

Returns

String[] The list of port names

Arguments

interface Specifies the interface name.

Example

```
get_component_interface_ports s0
```

Related Links

- get_component_interface_port_property on page 483
- get_component_interfaces on page 486
- load_component on page 494

get_component_interface_property

Description

Returns the value of a single property from the specified interface for the loaded component.

Usage

`get_component_interface_property` <*interface*> <*property*>

Returns

String The property value.

Arguments

interface Specifies the interface name.

property Specifies the property name. Refer to *Element Properties*.

Example

```
get_interface_property clk_in DISPLAY_NAME
```

Related Links

- load_component on page 494
- Element Properties on page 554
- get_interface_properties on page 517

get_component_interfaces

Description

Returns the list of interfaces in the loaded component.

Usage

```
get_component_interfaces
```

Returns

String[] The list of interface names.

Arguments

No arguments

Example

```
get_component_interfaces
```

Related Links

- get_component_interface_ports on page 484
- get_component_interface_property on page 485
- load_component on page 494

get_component_parameter_property

Description

Returns the property value of a parameter in the loaded component.

Usage

`get_component_parameter_property` <*parameter*> <*property*>

Returns

Various The parameter property value.

Arguments

parameter Specifies the parameter name in the component.

property Specifies the property name of the parameter. Refer to *Parameter Properties*.

Example

```
get_component_parameter_property baudRate ENABLED
```

Related Links

- get_component_parameters on page 489
- get_parameter_properties on page 542
- load_component on page 494
- Parameter Properties on page 559

get_component_parameter_value

Description

Returns the parameter value in the loaded component.

Usage

`get_component_parameter_value <parameter>`

Returns

various The parameter value

Arguments

parameter Specifies the parameter name in the component.

Example

```
get_component_parameter_value baudRate
```

Related Links

- get_component_parameters on page 489
- load_component on page 494
- set_component_parameter_value on page 497

get_component_parameters

Description

Returns the list of parameters in the loaded component.

Usage

```
get_component_parameters
```

Returns

String[] The list of parameters in the component.

Arguments

No arguments

Example

```
get_instance_parameters
```

Related Links

- get_component_parameter_property on page 487
- get_component_parameter_value on page 488
- load_component on page 494
- set_component_parameter_value on page 497

get_component_project_properties

Description

Returns the list of properties that you query about the loaded component's Quartus Prime project.

Usage

```
get_component_project_properties
```

Returns

String[] The list of project properties.

Arguments

No arguments

Example

```
get_component_project_properties
```

Related Links

- get_component_project_property on page 491
- load_component on page 494
- set_component_project_property on page 498

get_component_project_property

Description

Returns the project property value of the loaded component. Only select project properties are available.

Usage

`get_component_project_property <property>`

Returns

String The property value.

Arguments

property Specifies the project property name. Refer to *Project Properties*.

Example

```
get_component_project_property HIDE_FROM_IP_CATALOG
```

Related Links

- get_component_project_properties on page 490
- load_component on page 494
- set_component_project_property on page 498
- Project Properties on page 564

get_component_property

Description

Returns the property value of the loaded component.

Usage

`get_component_property <property>`

Returns

String The property value.

Arguments

property The property name on the loaded component. Refer to *Element Properties*.

Example

```
get_component_property CLASS_NAME
```

Related Links

- load_component on page 494
- get_instance_properties on page 426
- Element Properties on page 554

get_loaded_component

Description

Returns the instance name associated with the loaded component.

Usage

```
get_loaded_component
```

Returns

String The instance name.

Arguments

No arguments

Example

```
get_loaded_component
```

Related Links

- load_component on page 494
- save_component on page 496

load_component

Description

Loads the actual component inside of a generic component, so that you can modify the component parameters.

Usage

`load_component <instance>`

Returns

boolean 1 if successful; 0 if unsuccessful.

Arguments

instance Specifies the instance name.

Example

```
load_component cpu
```

Related Links

- get_loaded_component on page 493
- save_component on page 496

reload_component_footprint

Description

Validates the footprint of a specified child instance, and updates the footprint of the instance in case of issues.

Usage

`reload_component_footprint [<instance>]`

Returns

String[] A list of validation messages.

Arguments

instance (*optional*)	Specifies the child instance name to validate. If you do not specify this option, the command validates all the generic components in the system.

Example

```
reload_component_footprint cpu_0
```

Related Links

- load_instantiation on page 461
- save_instantiation on page 465
- validate_component_footprint on page 534

save_component

Description

Saves the loaded component.

Usage

```
save_component
```

Returns

No return value

Arguments

No arguments

Example

```
save_component
```

Related Links

- get_loaded_component on page 493
- load_component on page 494

set_component_parameter_value

Description

Sets the parameter value for the loaded component.

Usage

`set_component_parameter_value` <*parameter*> <*value*>

Returns

No return value

Arguments

parameter Specifies the parameter name.

parameter Specifies the new parameter value.

Example

```
set_component_parameter_value baudRate 9600
```

Related Links

- get_component_parameter_value on page 488
- get_component_parameters on page 489
- load_component on page 494

set_component_project_property

Description

Sets the project property value of the loaded component, such as hiding from the IP catalog.

Usage

`set_component_project_property` <*property*> <*value*>

Returns

No return value

Arguments

property Specifies the property name. Refer to *Project Properties*.

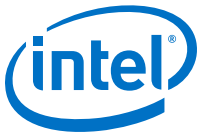
value Specifies the new property value.

Example

```
set_component_project_property HIDE_FROM_IP_CATALOG false
```

Related Links

- get_component_project_properties on page 490
- get_component_project_property on page 491
- load_component on page 494
- Project Properties on page 564



9.19.6.1.6 Connections

This section lists the commands that allow you to manipulate the interface connections in your Qsys Pro system.

add_connection

Description

Connects the named interfaces using an appropriate connection type. Both interface names consist of an instance name, followed by the interface name that the module provides.

Usage

`add_connection <start> [<end>]`

Returns

No return value.

Arguments

start The start interface that you connect, in *<instance_name>.<interface_name>* format. If you do not specify the end argument, the connection must be of the form *<instance1>.<interface>/<instance2>.<interface>*.

end (optional) The end interface that you connect, in *<instance_name>.<interface_name>* format.

Example

```
add_connection dma.read_master sdram.s1
```

Related Links

- get_connection_parameter_value on page 503
- get_connection_property on page 506
- get_connections on page 507
- remove_connection on page 508
- set_connection_parameter_value on page 510

auto_connect

Description

Creates connections from an instance or instance interface to matching interfaces of other instances in the system. For example, Avalon-MM slaves connect to Avalon-MM masters.

Usage

`auto_connect` <*element*>

Returns

No return value.

Arguments

element The instance interface name, or the instance name.

Example

```
auto_connect sdram
auto_connect uart_0.s1
```

Related Links

add_connection on page 500

get_connection_parameter_property

Description

Returns the property value of a parameter in a connection. Parameter properties are metadata about how Qsys Pro uses the parameter.

Usage

`get_connection_parameter_property` <*connection*> <*parameter*> <*property*>

Returns

various The parameter property value.

Arguments

connection The connection to query.

parameter The parameter name.

property The property of the connection. Refer to *Parameter Properties*.

Example

```
get_connection_parameter_property cpu.data_master/dma0.csr baseAddress UNITS
```

Related Links

- get_connection_parameter_value on page 503
- get_connection_property on page 506
- get_connections on page 507
- get_parameter_properties on page 542
- Parameter Properties on page 559

get_connection_parameter_value

Description

Returns the parameter value of the connection. Parameters represent aspects of the connection that you can modify, such as the base address for an Avalon-MM connection.

Usage

`get_connection_parameter_value` <*connection*> <*parameter*>

Returns

various The parameter value.

Arguments

connection The connection to query.

parameter The parameter name.

Example

```
get_connection_parameter_value cpu.data_master/dma0.csr baseAddress
```

Related Links

- get_connection_parameters on page 504
- get_connections on page 507
- set_connection_parameter_value on page 510

get_connection_parameters

Description

Returns the list of parameters of a connection.

Usage

`get_connection_parameters` <*connection*>

Returns

String[] The list of parameter names.

Arguments

connection The connection to query.

Example

```
get_connection_parameters cpu.data_master/dma0.csr
```

Related Links

- get_connection_parameter_property on page 502
- get_connection_parameter_value on page 503
- get_connection_property on page 506

get_connection_properties

Description

Returns the properties list of a connection.

Usage

```
get_connection_properties
```

Returns

String[] The list of connection properties.

Arguments

No arguments.

Example

```
get_connection_properties
```

Related Links

- get_connection_property on page 506
- get_connections on page 507

get_connection_property

Description

Returns the property value of a connection. Properties represent aspects of the connection that you can modify, such as the connection type.

Usage

`get_connection_property` <*connection*> <*property*>

Returns

String The connection property value.

Arguments

connection The connection to query.

property The connection property name. Refer to *Connection Properties*.

Example

```
get_connection_property cpu.data_master/dma0.csr TYPE
```

Related Links

- get_connection_properties on page 505
- Connection Properties on page 551

get_connections

Description

Returns the list of all connections in the system if you do not specify any element. If you specify a child instance, for example `cpu`, Qsys Pro returns all connections to any interface on the instance. If you specify an interface of a child instance, for example `cpu.instruction_master`, Qsys Pro returns all connections to that interface.

Usage

`get_connections` [<*element*>]

Returns

String[] The list of connections.

Arguments

element (optional) The child instance name, or the qualified interface name on a child instance.

Example

```
get_connections
get_connections cpu
get_connections cpu.instruction_master
```

Related Links

- add_connection on page 500
- remove_connection on page 508

remove_connection

Description

Removes a connection from the system.

Usage

`remove_connection` <*connection*>

Returns

No return value.

Arguments

connection The connection name to remove.

Example

```
remove_connection cpu.data_master/sdram.s0
```

Related Links

- add_connection on page 500
- get_connections on page 507

remove_dangling_connections

Description

Removes connections where both end points of the connection no longer exist in the system.

Usage

```
remove_dangling_connections
```

Returns

No return value.

Arguments

No arguments.

Example

```
remove_dangling_connections
```

Related Links

- add_connection on page 500
- get_connections on page 507
- remove_connection on page 508

set_connection_parameter_value

Description

Sets the parameter value for a connection.

Usage

`set_connection_parameter_value` <*connection*> <*parameter*> <*value*>

Returns

No return value.

Arguments

connection The connection name.

parameter The parameter name.

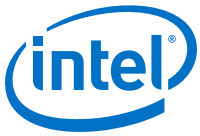
value The new parameter value.

Example

```
set_connection_parameter_value cpu.data_master/dma0.csr baseAddress "0x000a0000"
```

Related Links

- get_connection_parameter_value on page 503
- get_connection_parameters on page 504



9.19.6.1.7 Top-level Exports

This section lists the commands that allow you to manipulate the exported interfaces in your Qsys Pro system.

add_interface

Description

Adds an interface to your system, which Qsys Pro uses to export an interface from within the system. You specify the exported internal interface with `set_interface_property <interface> EXPORT_OF instance.interface`.

Usage

`add_interface <name> <type> <direction>`.

Returns

No return value.

Arguments

name The name of the interface that Qsys Pro exports from the system.

type The type of interface.

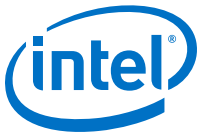
direction The interface direction.

Example

```
add_interface my_export conduit end
set_interface_property my_export EXPORT_OF uart_0.external_connection
```

Related Links

- get_interface_ports on page 516
- get_interface_properties on page 517
- get_interface_property on page 518
- set_interface_property on page 524



get_exported_interface_sysinfo_parameter_value

Description

Gets the value of a system info parameter for an exported interface.

Usage

`get_exported_interface_sysinfo_parameter_value` <*interface*> <*parameter*>

Returns

various The system info parameter value.

Arguments

interface Specifies the name of the exported interface.

parameter Specifies the name of the system info parameter. Refer to *System Info Type*.

Example

```
get_exported_interface_sysinfo_parameter_value clk clock_rate
```

Related Links

- get_exported_interface_sysinfo_parameters on page 514
- set_exported_interface_sysinfo_parameter_value on page 522
- System Info Type Properties on page 565

get_exported_interface_sysinfo_parameters

Description

Returns the list of system info parameters for an exported interface.

Usage

`get_exported_interface_sysinfo_parameters` <*interface*> [<*type*>]

Returns

String[] The list of system info parameter names.

Arguments

interface Specifies the name of the exported interface.

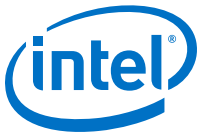
type (*optional*) Specifies the parameters type to return. If you do not specify this option, the command returns all the parameters. Refer to *Access Type*.

Example

```
get_exported_interface_sysinfo_parameters clk
```

Related Links

- get_exported_interface_sysinfo_parameter_value on page 513
- set_exported_interface_sysinfo_parameter_value on page 522
- Access Type on page 571



get_interface_port_property

Description

Returns the value of a property of a port contained by one of the top-level exported interfaces.

Usage

`get_interface_port_property` <*interface*> <*port*> <*property*>

Returns

various The property value.

Arguments

interface The name of a top-level interface of the system.

port The port name in the interface.

property The property name on the port. Refer to *Port Properties*.

Example

```
get_interface_port_property uart_exports tx DIRECTION
```

Related Links

- get_interface_ports on page 516
- get_port_properties on page 520
- Port Properties on page 563

get_interface_ports

Description

Returns the names of all the added ports to a given interface.

Usage

`get_interface_ports <interface>`

Returns

String[] The list of port names.

Arguments

interface The top-level interface name of the system.

Example

```
get_interface_ports export_clk_out
```

Related Links

- get_interface_port_property on page 515
- get_interfaces on page 519

get_interface_properties

Description

Returns the names of all the available interface properties common to all interface types.

Usage

```
get_interface_properties
```

Returns

String[] The list of interface properties.

Arguments

No arguments.

Example

```
get_interface_properties
```

Related Links

- get_interface_property on page 518
- set_interface_property on page 524

get_interface_property

Description

Returns the value of a single interface property from the specified interface.

Usage

`get_interface_property` <*interface*> <*property*>

Returns

various The property value.

Arguments

interface The name of a top-level interface of the system.

property The name of the property. Refer to *Interface Properties*.

Example

```
get_interface_property export_clk_out EXPORT_OF
```

Related Links

- get_interface_properties on page 517
- set_interface_property on page 524
- Interface Properties on page 556

get_interfaces

Description

Returns the list of top-level interfaces in the system.

Usage

```
get_interfaces
```

Returns

String[] The list of the top-level interfaces exported from the system.

Arguments

No arguments.

Example

```
get_interfaces
```

Related Links

- add_interface on page 512
- get_interface_ports on page 516
- get_interface_property on page 518
- remove_interface on page 521
- set_interface_property on page 524

get_port_properties

Description

Returns the list of properties that you can query for ports.

Usage

```
get_port_properties
```

Returns

String[] The list of port properties.

Arguments

No arguments.

Example

```
get_port_properties
```

Related Links

- get_instance_interface_port_property on page 416
- get_instance_interface_ports on page 417
- get_instance_port_property on page 425
- get_interface_port_property on page 515
- get_interface_ports on page 516

remove_interface

Description

Removes an exported top-level interface from the system.

Usage

`remove_interface <interface>`

Returns

No return value.

Arguments

interface The name of the exported top-level interface.

Example

```
remove_interface clk_out
```

Related Links

- add_interface on page 512
- get_interfaces on page 519

set_exported_interface_sysinfo_parameter_value

Description

Sets the system info parameter value for an exported interface.

Usage

`set_exported_interface_sysinfo_parameter_value` <*interface*> <*parameter*> <*value*>

Returns

No return value

Arguments

interface Specifies the name of the exported interface.

parameter Specifies the name of the system info parameter. Refer to *System Info Type*.

value Specifies the system info parameter value.

Example

```
set_exported_interface_sysinfo_parameter_value clk clock_rate 5000000
```

Related Links

- get_exported_interface_sysinfo_parameter_value on page 513
- get_exported_interface_sysinfo_parameters on page 514
- System Info Type Properties on page 565

set_interface_port_property

Description

Sets the port property in a top-level interface of the system.

Usage

`set_interface_port_property` <*interface*> <*port*> <*property*> <*value*>

Returns

No return value

Arguments

interface Specifies the top-level interface name of the system.

port Specifies the port name in a top-level interface of the system.

property Specifies the property name of the port. Refer to *Port Properties*.

value Specifies the property value.

Example

```
set_interface_port_property clk clk_clk NAME my_clk
```

Related Links

- Port Properties on page 576
- get_interface_ports on page 516
- get_interfaces on page 519
- get_port_properties on page 520

set_interface_property

Description

Sets the value of a property on an exported top-level interface. You use this command to set the EXPORT_OF property to specify which interface of a child instance is exported via this top-level interface.

Usage

`set_interface_property` <*interface*> <*property*> <*value*>

Returns

No return value.

Arguments

interface The name of an exported top-level interface.

property The name of the property. Refer to *Interface Properties*.

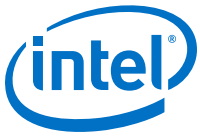
value The property value.

Example

```
set_interface_property clk_out EXPORT_OF clk.clk_out
```

Related Links

- add_interface on page 512
- get_interface_properties on page 517
- get_interface_property on page 518
- Interface Properties on page 556



9.19.6.1.8 Validation

This section lists the commands that allow you to validate the components, instances, interfaces and connections in your Qsys Pro system.

set_validation_property

Description

Sets a property that affects how and when validation is run. To disable system validation after each scripting command, set `AUTOMATIC_VALIDATION` to `False`.

Usage

`set_validation_property` <*property*> <*value*>

Returns

No return value.

Arguments

property The name of the property. Refer to *Validation Properties*.

value The new property value.

Example

```
set_validation_property AUTOMATIC_VALIDATION false
```

Related Links

- validate_system on page 533
- Validation Properties on page 568

sync_sysinfo_parameters

Description

Updates the system info parameters of the specified generic component.

Usage

`sync_sysinfo_parameters [<instance> ]`

Returns

String[] A list of update messages.

Arguments

instance (*optional*)	Specifies the name of the instance to sync. If you do not specify this option, the command synchronizes all the generic components in the system.

Example

```
sync_sysinfo_parameters cpu_0
```

Related Links

- load_instantiation on page 461
- save_instantiation on page 465

validate_component

Description

Validates the loaded component.

Usage

```
validate_component
```

Returns

String[] A list of validation messages.

Arguments

No arguments

Example

```
validate_component
```

Related Links

- validate_component_interface on page 529
- load_component on page 494

validate_component_interface

Description

Validates an interface of the loaded component.

Usage

`validate_component_interface` <*interface*>

Returns

String[] List of validation messages

Arguments

instance Specifies the name of the instance for the loaded component.

Example

```
validate_instance_interface data_master
```

Related Links

- load_component on page 494
- validate_component on page 528

validate_connection

Description

Validates the specified connection and returns validation messages.

Usage

`validate_connection <connection>`

Returns

A list of validation messages.

Arguments

connection The connection name to validate.

Example

```
validate_connection cpu.data_master/sdram.s1
```

Related Links

- validate_instance on page 531
- validate_instance_interface on page 532
- validate_system on page 533

validate_instance

Description

Validates the specified child instance and returns validation messages.

Usage

`validate_instance <instance>`

Returns

A list of validation messages.

Arguments

instance The child instance name to validate.

Example

```
validate_instance cpu
```

Related Links

- validate_connection on page 530
- validate_instance_interface on page 532
- validate_system on page 533

validate_instance_interface

Description

Validates an interface of an instance and returns validation messages.

Usage

`validate_instance_interface` <*instance*> <*interface*>

Returns

A list of validation messages.

Arguments

instance The child instance name.

interface The interface to validate.

Example

```
validate_instance_interface cpu data_master
```

Related Links

- validate_connection on page 530
- validate_instance on page 531
- validate_system on page 533

validate_system

Description

Validates the system and returns validation messages.

Usage

```
validate_system
```

Returns

A list of validation messages.

Arguments

No arguments.

Example

```
validate_system
```

Related Links

- validate_connection on page 530
- validate_instance on page 531
- validate_instance_interface on page 532

validate_component_footprint

Description

Validates the footprint of the specified child instance.

Usage

`validate_component_footprint <instance>`

Returns

String[] List of validation messages.

Arguments

instance (*optional*) Specifies the child instance name. If you omit this option, the command validates all generic components in the system.

Example

```
validate_component_footprint cpu_0
```

Related Links

- load_instantiation on page 461
- save_instantiation on page 465

reload_component_footprint

Description

Validates the footprint of a specified child instance, and updates the footprint of the instance in case of issues.

Usage

`reload_component_footprint` [<*instance*>]

Returns

String[] A list of validation messages.

Arguments

instance (*optional*)	Specifies the child instance name to validate. If you do not specify this option, the command validates all the generic components in the system.

Example

```
reload_component_footprint cpu_0
```

Related Links

- load_instantiation on page 461
- save_instantiation on page 465
- validate_component_footprint on page 534

9.19.6.1.9 Miscellaneous

This section lists the miscellaneous commands that you can use for your Qsys Pro systems.

auto_assign_base_addresses

Description

Assigns base addresses to all memory-mapped interfaces of an instance in the system. Instance interfaces that are locked with `lock_avalon_base_address` keep their addresses during address auto-assignment.

Usage

`auto_assign_base_addresses` <*instance*>

Returns

No return value.

Arguments

instance The name of the instance with memory-mapped interfaces.

Example

```
auto_assign_base_addresses sdram
```

Related Links

- auto_assign_system_base_addresses on page 539
- lock_avalon_base_address on page 543
- unlock_avalon_base_address on page 547

auto_assign_irqs

Description

Assigns interrupt numbers to all connected interrupt senders of an instance in the system.

Usage

`auto_assign_irqs <instance>`

Returns

No return value.

Arguments

instance The name of the instance with an interrupt sender.

Example

```
auto_assign_irqs uart_0
```

auto_assign_system_base_addresses

Description

Assigns legal base addresses to all memory-mapped interfaces of all instances in the system. Instance interfaces that are locked with `lock_avalon_base_address` keep their addresses during address auto-assignment.

Usage

```
auto_assign_system_base_addresses
```

Returns

No return value.

Arguments

No arguments.

Example

```
auto_assign_system_base_addresses
```

Related Links

- auto_assign_base_addresses on page 537
- lock_avalon_base_address on page 543
- unlock_avalon_base_address on page 547

get_interconnect_requirement

Description

Returns the value of an interconnect requirement for a system or interface of a child instance.

Usage

`get_interconnect_requirement` <*element_id*> <*requirement*>

Returns

String The value of the interconnect requirement.

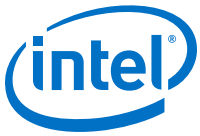
Arguments

element_id {$system} for the system, or the qualified name of the interface of an instance, in <*instance*>.<*interface*> format. In Tcl, the system identifier is escaped, for example, {$system}.

requirement The name of the requirement.

Example

```
get_interconnect_requirement {$system} qsys_mm.maxAdditionalLatency
```



get_interconnect_requirements

Description

Returns the list of all interconnect requirements in the system.

Usage

```
get_interconnect_requirements
```

Returns

String[] A flattened list of interconnect requirements. Every sequence of three elements in the list corresponds to one interconnect requirement. The first element in the sequence is the element identifier. The second element is the requirement name. The third element is the value. You can loop over the returned list with a `foreach loop`, for example:

```
foreach { element_id name value } $requirement_list { loop_body
          }
```

Arguments

No arguments.

Example

```
get_interconnect_requirements
```

get_parameter_properties

Description

Returns the list of properties that you can query for any parameters, for example parameters of instances, interfaces, instance interfaces, and connections.

Usage

```
get_parameter_properties
```

Returns

String[] The list of parameter properties.

Arguments

No arguments.

Example

```
get_parameter_properties
```

Related Links

- get_connection_parameter_property on page 502
- get_instance_interface_parameter_property on page 413
- get_instance_parameter_property on page 421

lock_avalon_base_address

Description

Prevents the memory-mapped base address from being changed for connections to the specified interface of an instance when Qsys Pro runs the `auto_assign_base_addresses` or `auto_assign_system_base_addresses` commands.

Usage

`lock_avalon_base_address` <*instance.interface*>

Returns

No return value.

Arguments

instance.interface	The qualified name of the interface of an instance, in <*instance*>.<*interface*> format.

Example

```
lock_avalon_base_address sdram.s1
```

Related Links

- auto_assign_base_addresses on page 537
- auto_assign_system_base_addresses on page 539
- unlock_avalon_base_address on page 547

send_message

Description

Sends a message to the user of the component. The message text is normally HTML. You can use the <*b*> element to provide emphasis. If you do not want the message text to be HTML, then pass a list like `{ Info Text }` as the message level,

Usage

`send_message` <*level*> <*message*>

Returns

No return value.

Arguments

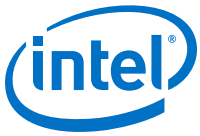
level Quartus Prime supports the following message levels:

- `ERROR`—provides an error message.
- `WARNING`—provides a warning message.
- `INFO`—provides an informational message.
- `PROGRESS`—provides a progress message.
- `DEBUG`—provides a debug message when debug mode is enabled.

message The text of the message.

Example

```
send_message ERROR "The system is down!"
send_message { Info Text } "The system is up!"
```



set_interconnect_requirement

Description

Sets the value of an interconnect requirement for a system or an interface of a child instance.

Usage

`set_interconnect_requirement` <*element_id*> <*requirement*> <*value*>

Returns

No return value.

Arguments

element_id `{$system}` for the system, or qualified name of the interface of an instance, in <*instance*>.<*interface*> format. In Tcl, the system identifier is escaped, for example, `{$system}`.

requirement The name of the requirement.

value The requirement value.

Example

```
set_interconnect_requirement {$system} qsys_mm.clockCrossingAdapter HANDSHAKE
```

set_use_testbench_naming_pattern

Description

Use this command to create testbench systems so that the generated file names for the test system match the system's original generated file names. Without setting this command, the generated file names for the test system receive the top-level testbench system name.

Usage

`set_use_testbench_naming_pattern` <*value*>

Returns

No return value.

Arguments

value True or false.

Example

```
set_use_testbench_naming_pattern true
```

Notes

Use this command only to create testbench systems.

unlock_avalon_base_address

Description

Allows the memory-mapped base address to change for connections to the specified interface of an instance when Qsys Pro runs the `auto_assign_base_addresses` or `auto_assign_system_base_addresses` commands.

Usage

`unlock_avalon_base_address` <*instance.interface*>

Returns

No return value.

Arguments

instance.interface	The qualified name of the interface of an instance, in <*instance*>.<*interface*> format.

Example

```
unlock_avalon_base_address sdram.s1
```

Related Links

- auto_assign_base_addresses on page 537
- auto_assign_system_base_addresses on page 539
- lock_avalon_base_address on page 543

get_testbench_dutname

Description

Returns the currently set dutname for the test-bench systems. Use this command only when creating test-bench systems.

Usage

```
get_testbench_dutname
```

Returns

String The currently set dutname. Returns NULL if empty.

Arguments

No arguments.

Example

```
get_testbench_dutname
```

Related Links

- get_use_testbench_naming_pattern on page 549
- set_use_testbench_naming_pattern on page 546

get_use_testbench_naming_pattern

Description

Verifies if the test-bench naming pattern is set to be used. Use this command only when creating test-bench systems.

Usage

```
get_use_testbench_naming_pattern
```

Returns

boolean True, if the test-bench naming pattern is set to be used.

Arguments

No arguments.

Example

```
get_use_testbench_naming_pattern
```

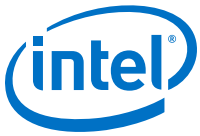
Related Links

- get_testbench_dutname on page 548
- set_use_testbench_naming_pattern on page 546

9.19.6.2 Qsys Pro Scripting Property Reference

Interface properties work differently for **_hw.tcl** scripting than with Qsys Pro scripting. In **_hw.tcl**, interfaces do not distinguish between properties and parameters. In Qsys Pro scripting, the properties and parameters are unique.

The following are the Qsys Pro scripting properties:



9.19.6.2.1 Connection Properties

Type	Name	Description
string	END	Indicates the end interface of the connection.
string	NAME	Indicates the name of the connection.
string	START	Indicates the start interface of the connection.
String	TYPE	The type of the connection.

9.19.6.2.2 Design Environment Type Properties

Description

IP cores use the design environment to identify the most appropriate interfaces to connect to the parent system.

Name	Description
NATIVE	Supports native IP interfaces.
QSYS	Supports standard Qsys Pro interfaces.

9.19.6.2.3 Direction Properties

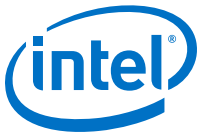
Name	Description
BIDIR	Indicates the direction for a bidirectional signal.
INOUT	Indicates the direction for an input signal.
OUTPUT	Indicates the direction for an output signal.

9.19.6.2.4 Element Properties

Description

Element properties are, with the exception of ENABLED and NAME, read-only properties of the types of instances, interfaces, and connections. These read-only properties represent metadata that does not vary between copies of the same type. ENABLED and NAME properties are specific to particular instances, interfaces, or connections.

Type	Name	Description
String	AUTHOR	The author of the component or interface.
Boolean	AUTO_EXPORT	Indicates whether unconnected interfaces on the instance are automatically exported.
String	CLASS_NAME	The type of the instance, interface or connection, for example, altera_nios2 or avalon_slave.
String	DESCRIPTION	The description of the instance, interface or connection type.
String	DISPLAY_NAME	The display name for referencing the type of instance, interface or connection.
Boolean	EDITABLE	Indicates whether you can edit the component in the Qsys Pro Component Editor.
Boolean	ENABLED	Indicates whether the instance is enabled.
String	GROUP	The IP Catalog category.
Boolean	INTERNAL	Hides internal IP components or sub-components from the IP Catalog..
String	NAME	The name of the instance, interface or connection.
String	VERSION	The version number of the instance, interface or connection, for example, 16.1.

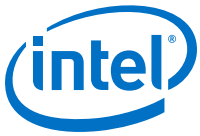


9.19.6.2.5 Instance Properties

Type	Name	Description
String	AUTO_EXPORT	Indicates whether Qsys Pro automatically exports the unconnected interfaces on the instance.
Boolean	ENABLED	If true, Qsys Pro includes this instance in the generated system.
String	NAME	The name of the system, which Qsys Pro uses as the name of the top-level module in the generated HDL.

9.19.6.2.6 Interface Properties

Type	Name	Description
String	EXPORT_OF	Indicates which interface of a child instance to export through the top-level interface. Before using this command, you must create the top-level interface using the `add_interface` command. You must use the format: *<instanceName.interfaceName>*. For example: `set_interface_property CSC_input EXPORT_OF my_colorSpaceConverter.input_port`



9.19.6.2.7 Message Levels Properties

Name	Description
COMPONENT_INFO	Reports an informational message only during component editing.
DEBUG	Provides messages when debug mode is enabled.
ERROR	Provides an error message.
INFO	Provides an informational message.
PROGRESS	Reports progress during generation.
TODOERROR	Provides an error message that indicates the system is incomplete.
WARNING	Provides a warning message.

9.19.6.2.8 Module Properties

Type	Name	Description
String	GENERATION_ID	The generation ID for the system.
String	NAME	The name of the instance.

9.19.6.2.9 Parameter Properties

Type	Name	Description
Boolean	AFFECTS_ELABORATION	Set AFFECTS_ELABORATION to false for parameters that do not affect the external interface of the module. An example of a parameter that does not affect the external interface is isNonVolatileStorage. An example of a parameter that does affect the external interface is width. When the value of a parameter changes and AFFECTS_ELABORATION is false, the elaboration phase does not repeat and improves performance. When AFFECTS_ELABORATION is set to true, the default value, Qsys Pro reanalyzes the HDL file to determine the port widths and configuration each time a parameter changes.
Boolean	AFFECTS_GENERATION	The default value of AFFECTS_GENERATION is false if you provide a top-level HDL module. The default value is true if you provide a fileset callback. Set AFFECTS_GENERATION to false if the value of a parameter does not change the results of fileset generation.
Boolean	AFFECTS_VALIDATION	The AFFECTS_VALIDATION property determines whether a parameter's value sets derived parameters, and whether the value affects validation messages. Setting this property to false may improve response time in the parameter editor when the value changes.
String[]	ALLOWED_RANGES	Indicates the range or ranges of the parameter. For integers, each range is a single value, or a range of values defined by a start and end value, and delimited by a colon, for example, 11:15. This property also specifies the legal values and description strings for integers, for example, {0:None 1:Monophonic 2:Stereo 4:Quadrophonic}, where 0, 1, 2, and 4 are the legal values. You can assign description strings in the parameter editor for string variables. For example, `ALLOWED_RANGES {"dev1:Cyclone IV GX""dev2:Stratix V GT"}`
String	DEFAULT_VALUE	The default value.
Boolean	DERIVED	When True, indicates that the parameter value is set by the component and cannot be set by the user. Derived parameters are not saved as part of an instance's parameter values. The default value is False.
String	DESCRIPTION	A short user-visible description of the parameter, suitable for a tooltip description in the parameter editor.
String[]	DISPLAY_HINT	Provides a hint about how to display a property. • boolean--For integer parameters whose value are 0 or 1. The parameter displays as an option that you can turn on or off. • radio—displays a parameter with a list of values as radio buttons. • hexadecimal—for integer parameters, displays and interprets the value as a hexadecimal number, for example: 0x00000010 instead of 16. • fixed_size—for string_list and integer_list parameters, the fixed_size DISPLAY_HINT eliminates the **Add** and **Remove** buttons from tables.
String	DISPLAY_NAME	The GUI label that appears to the left of this parameter.
String	DISPLAY_UNITS	The GUI label that appears to the right of the parameter.
Boolean	ENABLED	When False, the parameter is disabled. The parameter displays in the parameter editor but is grayed out, indicating that you cannot edit this parameter.

Type	Name	Description
String	GROUP	Controls the layout of parameters in the GUI.
Boolean	HDL_PARAMETER	When True, Qsys Pro passes the parameter to the HDL component description. The default value is False.
String	LONG_DESCRIPTION	A user-visible description of the parameter. Similar to DESCRIPTION, but allows a more detailed explanation.
String	NEW_INSTANCE_VALUE	Changes the default value of a parameter without affecting older components that do not explicitly set a parameter value, and use the DEFAULT_VALUE property. Oder instances continue to use DEFAULT_VALUE for the parameter and new instances use the value assigned by NEW_INSTANCE_VALUE.
String[]	SYSTEM_INFO	Allows you to assign information about the instantiating system to a parameter that you define. SYSTEM_INFO requires an argument specifying the type of information. For example: `SYSTEM_INFO <info-type>`
String	SYSTEM_INFO_ARG	Defines an argument to pass to SYSTEM_INFO. For example, the name of a reset interface.
(various)	SYSTEM_INFO_TYPE	Specifies the types of system information that you can query. Refer to *System Info Type Properties*.
(various)	TYPE	Specifies the type of the parameter. Refer to *Parameter Type Properties*.
(various)	UNITS	Sets the units of the parameter. Refer to *Units Properties*.
Boolean	VISIBLE	Indicates whether or not to display the parameter in the parameter editor.
String	WIDTH	Indicates the width of the logic vector for the STD_LOGIC_VECTOR parameter.

Related Links

- System Info Type Properties on page 565
- Parameter Type Properties on page 562
- Units Properties on page 567

9.19.6.2.10 Parameter Status Properties

Type	Name	Description
Boolean	ACTIVE	Indicates that this parameter is an active parameter.
Boolean	DEPRECATED	Indicates that this parameter exists only for backwards compatibility, and may not have any effect.
Boolean	EXPERIMENTAL	Indicates that this parameter is experimental and not exposed in the design flow.

9.19.6.2.11 Parameter Type Properties

Name	Description
BOOLEAN	A boolean parameter set to `true` or `false`.
FLOAT	A signed 32-bit floating point parameter. (Not supported for HDL parameters.)
INTEGER	A signed 32-bit integer parameter.
INTEGER_LIST	A parameter that contains a list of 32-bit integers. (Not supported for HDL parameters.)
LONG	A signed 64-bit integer parameter. (Not supported for HDL parameters.)
NATURAL	A 32-bit number that contains values `0` to `2147483647` (`0x7fffffff`).
POSITIVE	A 32-bit number that contains values `1` to `2147483647` (`0x7fffffff`).
STD_LOGIC	A single bit parameter set to `0` or `1`.
STD_LOGIC_VECTOR	An arbitrary-width number. The parameter property `WIDTH` determines the size of the logic vector.
STRING	A string parameter.
STRING_LIST	A parameter that contains a list of strings. (Not supported for HDL parameters.)

9.19.6.2.12 Port Properties

Type	Name	Description
(various)	DIRECTION	The direction of the signal. Refer to *Direction Properties*.
String	ROLE	The type of the signal. Each interface type defines a set of interface types for its ports.
Integer	WIDTH	The width of the signal in bits.

Related Links

Direction Properties on page 553

9.19.6.2.13 Project Properties

Type	Name	Description
String	DEVICE	The device part number in the Quartus Prime project that contains the Qsys Pro system.
String	DEVICE_FAMILY	The device family name in the Quartus Prime project that contains the Qsys Pro system.

9.19.6.2.14 System Info Type Properties

Type	Name	Description
String	ADDRESS_MAP	An XML-formatted string that describes the address map for the interface specified in the SYSTEM_INFO parameter property.
Integer	ADDRESS_WIDTH	The number of address bits that Qsys Pro requires to address memory-mapped slaves connected to the specified memory-mapped master in this instance.
String	AVALON_SPEC	The version of the Qsys Pro interconnect. Refer to *Avalon Interface Specifications*.
Integer	CLOCK_DOMAIN	An integer that represents the clock domain for the interface specified in the SYSTEM_INFO parameter property. If this instance has interfaces on multiple clock domains, you can use this property to determine which interfaces are on each clock domain. The absolute value of the integer is arbitrary.
Long, Integer	CLOCK_RATE	The rate of the clock connected to the clock input specified in the SYSTEM_INFO parameter property. If zero, the clock rate is currently unknown.
String	CLOCK_RESET_INFO	The name of this instance's primary clock or reset sink interface. You use this property to determine the reset sink for global reset when you use SOPC Builder interconnect that conforms to *Avalon Interface Specifications*.
String	CUSTOM_INSTRUCTION_SLAVES	Provides slave information, including the name, base address, address span, and clock cycle type.
String	DESIGN_ENVIRONMENT	A string that identifies the current design environment. Refer to *Design Environment Type Properties*.
String	DEVICE	The device part number of the selected device.
String	DEVICE_FAMILY	The family name of the selected device.
String	DEVICE_FEATURES	A list of key/value pairs delimited by spaces that indicate whether a device feature is available in the selected device family. The format of the list is suitable for passing to the `array` command. The keys are device features. The values are 1 if the feature is present, and 0 if the feature is absent.
String	DEVICE_SPEEDGRADE	The speed grade of the selected device.
Integer	GENERATION_ID	An integer that stores a hash of the generation time that Qsys Pro uses as a unique ID for a generation run.
BigInteger, Long	INTERRUPTS_USED	A mask indicating which bits of an interrupt receiver are connected to interrupt senders. The interrupt receiver is specified in the system info argument.
Integer	MAX_SLAVE_DATA_WIDTH	The data width of the widest slave connected to the specified memory-mapped master.
String, Boolean, Integer	QUARTUS_INI	The value of the `quartus.ini` setting specified in the system info argument.
Integer	RESET_DOMAIN	An integer representing the reset domain for the interface specified in the SYSTEM_INFO parameter property If this instance has interfaces on multiple reset domains, you can use this property to determine which interfaces are on each reset domain. The absolute value of the integer is arbitrary.

Type	Name	Description
String	TRISTATECONDUIT_INFO	An XML description of the tri-state conduit masters connected to a tri-state conduit slave. The slave is specified as the SYSTEM_INFO parameter property. The value contains information about the slave, connected master instance and interface names, and signal names, directions, and widths.
String	TRISTATECONDUIT_MASTERS	The names of the instance's interfaces that are tri-state conduit slaves.
String	UNIQUE_ID	A string guaranteed to be unique to this instance.

Related Links

- Design Environment Type Properties on page 552
- Avalon Interface Specifications
- Qsys Pro Interconnect on page 627
  Qsys Pro interconnect is a high-bandwidth structure that allows you to connect IP components to other IP components with various interfaces.

9.19.6.2.15 Units Properties

Name	Description
ADDRESS	A memory-mapped address.
BITS	Memory size in bits.
BITSPERSECOND	Rate in bits per second.
BYTES	Memory size in bytes.
CYCLES	A latency or count in clock cycles.
GIGABITSPERSECOND	Rate in gigabits per second.
GIGABYTES	Memory size in gigabytes.
GIGAHERTZ	Frequency in GHz.
HERTZ	Frequency in Hz.
KILOBITSPERSECOND	Rate in kilobits per second.
KILOBYTES	Memory size in kilobytes.
KILOHERTZ	Frequency in kHz.
MEGABITSPERSECOND	Rate, in megabits per second.
MEGABYTES	Memory size in megabytes.
MEGAHERTZ	Frequency in MHz.
MICROSECONDS	Time in microseconds.
MILLISECONDS	Time in milliseconds.
NANOSECONDS	Time in nanoseconds.
NONE	Unspecified units.
PERCENT	A percentage.
PICOSECONDS	Time in picoseconds.
SECONDS	Time in seconds.

9.19.6.2.16 Validation Properties

Type	Name	Description
Boolean	AUTOMATIC_VALIDATION	When `true`, Qsys Pro runs system validation and elaboration after each scripting command. When `false`, Qsys Pro runs system validation with validation scripting commands. Some queries affected by system elaboration may be incorrect if automatic validation is disabled. You can disable validation to make a system script run faster.

9.19.6.2.17 Interface Direction

Type	Name	Description
String	INPUT	Indicates that the interface is a slave (input, transmitter, sink, or end).
String	OUTPUT	Indicates that the interface is a master (output, receiver, source, or start).

9.19.6.2.18 File Set Kind

Name	Description
EXAMPLE_DESIGN	This file-set contains example design files.
QUARTUS_SYNTH	This file-set contains files that Qsys Pro uses for Quartus Prime Synthesis
SIM_VERILOG	This file-set contains files that Qsys Pro uses for Verilog HDL Simulation.
SIM_VHDL	This file-set contains files that Qsys Pro uses for VHDL Simulation.

9.19.6.2.19 Access Type

Name	Type	Description
String	READ_ONLY	Indicates that the parameter can be only read-only.
String	WRITABLE	Indicates that the parameter has read/write properties.

9.19.6.2.20 Instantiation Hdl File Properties

Name	Type	Description
Boolean	CONTAINS_INLINE_CONFIGURATION	Returns *True* if the HDL file contains inline configuration.
Boolean	IS_CONFIGURATION_PACKAGE	Returns *True* if the HDL file is a configuration package.
Boolean	IS_TOP_LEVEL	Returns *True* if the HDL file is the top-level HDL file.
String	OUTPUT_PATH	Specifies the output path of the HDL file.
String	TYPE	Specifies the HDL file type of the HDL file.

9.19.6.2.21 Instantiation Interface Duplicate Type

Type	Name	Description
String	CLONE	Creates a copy of an interface and all the interface ports.
String	MIRROR	Creates a copy of an interface with all the port roles and directions reversed.

9.19.6.2.22 Instantiation Interface Properties

Name	Type	Description
String	DIRECTION	The direction of the interface.
String	TYPE	The type of the interface.

9.19.6.2.23 Instantiation Properties

Name	Type	Description
String	HDL_COMPILATION_LIBRARY	Indicates the HDL compilation library name of the generic component.
String	HDL_ENTITY_NAME	Indicates the HDL entity name of the Generic Component.
String	IP_FILE	Indicates the .ip file path that implements the generic component.

9.19.6.2.24 Port Properties

Name	Type	Description
Direction	DIRECTION	Specifies the direction of the signal
String	NAME	Renames a top-level port. Only use with set_interface_port_property
String	ROLE	Specifies the type of the signal. Each interface type defines a set of interface types for its ports.
String	VHDL_TYPE	Specifies the VHDL type of the signal. Can be either *STANDARD_LOGIC*, or *STANDARD_LOGIC_VECTOR*.
Integer	WIDTH	Specifies the width of the signal in bits.

9.19.6.2.25 VHDL Type

Name	Description
STD_LOGIC	Represents the value of a digital signal in a wire.
STD_LOGIC_VECTOR	Represents an array of digital signals and variables.

9.19.6.3 Parameterizing an Instantiated IP Core after save_system Command

When you call the `save_system` command in your Tcl script, Qsys Pro converts all the instantiated IP cores in your system to generic components.

To modify these IP cores after saving your system, you must first load the actual component within the instantiated generic component. Re-parameterize an instantiated IP core using one of the following methods:

1. Load the component in the Qsys Pro system, modify the component's parameter value, and save the component:

```
…
save_system kernel_system.qsys
…
load_component cra_root
set_component_parameter_value DATA_W 64
save_component
…
```

2. Load the `.ip` file specific to the component, modify the instance's parameter value, and save the `.ip` file:

```
…
save_system kernel_system.qsys
…
load_system cra_root.ip
set_instance_parameter_value cra_root DATA_W 64
save_system
…
```

Note: To directly modify an instance parameter value after the `save_system` command, you must load the `.ip` file corresponding to the IP component.

Related Links

- set_component_parameter_value on page 497
- load_component on page 494
- save_component on page 496
- save_system on page 389

9.19.7 Validate the Generic Components in a System with qsys-validate

Use the `qsys-validate` utility to run IP component footprint validation on the `.qsys` file for the system.

Table 108. qsys-validate Command-Line Options

Option	Usage	Description
1st arg file	Optional	The name of the **.qsys** system file to validate.
`--search-path[=<value>]`	Optional	If omitted, Qsys Pro uses a standard default path. If provided, Qsys Pro searches a comma-separated list of paths. To include the standard path in your replacement, use "$", for example: `/extra/dir,$`.
`--strict`	Optional	Enables strict validation. All warnings are reported as errors
`--jvm-max-heap-size=<value>`	Optional	The maximum memory size Qsys Pro uses for allocations when running `qsys-edit`. You specify this value as `<size><unit>`, where unit is `m` (or `M`) for multiples of megabytes, or `g` (or `G`) for multiples of gigabytes. The default value is `512m`.
`--help`	Optional	Display help for `qsys-validate`.

9.19.8 Archive a Qsys Pro System with qsys-archive

The `qsys-archive` command allows you to archive a system, extract an archived system, and retrieve information about the system's dependencies.

Table 109. `qsys-archive` Command-Line Options

Option	Usage	Description
<1st arg file>	Required	The filename of the root Qsys Pro system, Qsys Pro file archive, or the Quartus Prime project file.
`--search-path[=<value>]`	Optional	If you omit this option, Qsys Pro uses a standard default path. If you specify this option, Qsys Pro searches a comma-separated list of paths. To include the standard path in your replacement, use "$", for example: `/extra/dir,$`.
`--archive`	Optional	Creates a zip archive of the specified Qsys Pro system or the Quartus Prime project.
`--report-file[=<value>]`	Optional	Lists the files that the Qsys Pro system or the Quartus Prime project references, and writes the files list to the specified name in `.txt` format.
`--output-directory[=<file>]`	Optional	Specifies the output directory to save the archive.
`--extract`	Optional	Extracts all the files in the given archive.
`--output-name[=<value>]`	Optional	Specifies the output name to save the archive and/or report.
`collect-to-common-directory[=<true|false>]`	Optional	When archiving, collects all the `.qsys` files in the root directory of the archive and all `.ip` files in a single `ip` directory, and updates all the matching references. The default option is *true*.

continued...

Option	Usage	Description
`new-quartus-project[=<value>]`	Optional	Creates a new Quartus Prime project which contains all the `.ip` and system files referenced by the Qsys Pro system or the Quartus Prime project.
`quartus-project[=<value>]`	Optional	When you use this command in combination with: • `--report-file`—adds all the referenced files to the Quartus Prime project. • `--extract`—adds all extracted files to the specified project. • `--archive`—archives all the system and `.ip` files referenced in the Quartus Prime project.
`--rev`	Optional	Specifies the name of the Quartus Prime project revision.
`--include-generated-files`	Optional	Includes all the generated files of the Qsys Pro system.
`--force`	Optional	Forcefully creates the specified archive or report, overwriting any existing archives or reports.
`--jvm-max-heap-size=<value>`	Optional	Specifies the maximum memory size Qsys Pro uses for allocations when running `qsys-edit`. Specify this value as `<size><unit>`, where unit is `m` (or `M`) for multiples of megabytes, or `g` (or `G`) for multiples of gigabytes. The default value is `512m`.
`--help`	Optional	Displays help for `qsys-archive`.

Alternatively, you can archive/restore your system using the Qsys Pro GUI. For more information, refer to *Archive your System* section.

Related Links

Archive your System on page 328
Qsys Pro allows you to archive your system in a `.zip` format.

9.19.9 Generate an IP Component or Qsys Pro System with quartus_ipgenerate

The `quartus_ipgenerate` command allows you to generate IP components or a Qsys Pro system in your Quartus Prime project. Ensure that you include the IP component or the Qsys Pro system you wish to generate in your Quartus Prime project.

To run the `quartus_ipgenerate` command from the Quartus Prime shell, type:

```
quartus_ipgenerate <project name> [<options>]
```

Use any of the following options with the `quartus_ipgenerate` utility:

Table 110. `quartus_ipgenerate` Command-Line Options

Option	Usage	Description
<1st arg file>	Required	Specifies the name of the Quartus Prime project file (`.qpf`). This option generates all the `.qsys` and `.ip` files in the specified Quartus Prime project (*<project name>*).
`-f [`*<argument file>*`]`	Optional	Specifies a file containing additional command-line arguments. Arguments that you specify after this option can conflict or override the options you specify in the argument file.
`--rev[=`*<revision name>*`]` or `-c[=`*<revision name>*`]`	Optional	Specifies the Quartus Prime project revision and the associated `.qsf` file to use. If you omit this option, Qsys Pro uses the same revision name as your Quartus Prime project.

`--clear_ip_generation_dirs` or `--clean`	Optional	Clears the generation directories of all the `.qsys` or the `.ip` files in the specified Quartus Prime project. For example, to clear the generation directories in the project `test`, run the following command:
		`quartus_ipgenerate --clear_ip_generation_dirs test`
		or
		`quartus_ipgenerate --clean test`

`--generate_ip_file --ip_file[=`*<ip file name>*`]`	Optional	Generates the files for *<file name>*`.ip` file in the specified Quartus Prime project. Use the following optional flags with `--generate_ip_file`: • `-synthesis[=`*<value>*`]`—optional argument that specifies the synthesis target type. Specify the value as either *verilog* or *vhdl*. The default value is *verilog*. • `-simulation[=`*<value>*`]`—optional argument that specifies the simulation target type. Specify the value as either *verilog* or *vhdl*. If you omit this flag, Qsys Pro does not generate any simulation files. • `--clear_ip_generation_dirs`—clears the preexisting generation directories before generation. If you omit this command, Qsys Pro does not clear the generation directories. For example, to generate the files for a `test.qsys` file within the project, `test`: `quartus_ipgenerate --generate_ip_file --synthesis=vhdl --simulation=verilog --clear_ip_generation_dirs --ip_file=test.qsys test`
`--generate_project_ip_files [`*<project name>*`]`	Optional	Generates the files for all the `.qsys` and `.ip` files in the specified Quartus Prime project. Use any of the following optional flags with `--generate_project_ip_files`: • `-synthesis[=`*<value>*`]`—optional argument that specifies the synthesis target type. Specify the value as either *verilog* or *vhdl*. The default value is *verilog*. • `-simulation[=`*<value>*`]`—optional argument that specifies the simulation target type. Specify the value as either *verilog* or *vhdl*. If you omit this flag, Qsys Pro does not generate any simulation files. • `--clear_ip_generation_dirs`—clears the preexisting generation directories before generation. If you omit this command, Qsys Pro does not clear the generation directories.

continued...

Option	Usage	Description
		For example, to generate all the `.qsys` and `.ip` files within the project, `test`: `quartus_ipgenerate --generate_project_ip_files --synthesis=vhdl --simulation=verilog --clear_ip_generation_dirs test`
`--get_project_ip_files`	Optional	Returns a list of the `.qsys` or `.ip` files in the specified Quartus Prime project. This option displays each file in a separate Quartus Prime message line. For example, to get a list of `.qsys` files in the project `test`, and revision `rev`: `quartus_ipgenerate --get_project_ip_files test -c rev`
`--lower_priority`	Optional	Allows you to lower the priority of the current process. This option is useful if you use a single-processor computer, allowing you to use other applications more easily while the Quartus Prime software runs the command in the background.

9.19.10 Generate an IP Variation File with ip-deploy

Use the `ip-deploy` utility to generate an IP variation file (`.ip` file) in the specified location.

Table 111. ip-deploy Command-Line Options

Option	Usage	Description
`--component-name[=<value>]`	Required	The name of a component you instantiate.
`--output-name[=<value>]`	Optional	Name for the resulting component; defaults to the component's type name.
`--component-parameter[=<value>]`	Optional	Repeatable. A single value assignment, like `--component-param=WIDTH=11`. To assign multiple parameters, use this option several times.
`--preset[=<value>]`	Optional	Repeatable. The name of a saved preset to use in creating a variation of the IP component. Presets are additive and repeatable.
`--family[=<value>]`	Optional	Sets the device family
`--part[=<value>]`	Optional	Sets the device part number. You can also use this command to set the base device, device speed-grade, device family, and device feature's system information.
`--output-directory[=<value>]`	Optional	This directory contains the output IP variation file. Qsys Pro automatically creates the directory if the directory does not exist. If you don't specify an output directory, the output directory is the current working directory.

continued...

Option	Usage	Description
`--search-path[=<value>]`	Optional	If you do not specify the search path, a standard default path will be used. If you provide a search path, Qsys Pro searches a comma-separated list of paths. To include the standard path in your replacement, use "$", like `/extra/dir,$`.
`--jvm-max-heap-size[=<value>]`		The maximum memory size Qsys Pro uses for allocations when running `qsys-edit`. You specify this value as `<size><unit>`, where unit is `m` (or `M`) for multiples of megabytes, or `g` (or `G`) for multiples of gigabytes. The default value is `512m`.
`--help`	Optional	Displays help for `ip-deploy`

9.20 Document Revision History

The table below indicates edits made to the *Creating a System With Qsys Pro* content since its creation.

Table 112. Document Revision History

Date	Version	Changes
2017.05.06	17.0.0	• Updated the topic - *Create/Open Project in Qsys Pro* • Updated the topic - *Modify the Target Device* • Updated the topic - *Modify the IP Search Path* • Added new topic - *Save your System* • Added new topic - *Archive your System* • Added new topic - *Synchronize IP File References* • Updated the topic - *Upgrade Outdated IP Components in Qsys Pro* • Added new topic - *Run System Scripts* • Added new topic - *View Avalon Memory Mapped Domains in Your Qsys Pro System* • Updated the topic - *Qsys Pro Scripting Command Reference* for new Tcl scripting commands • Updated the topic - *Qsys Pro Scripting Property Reference* for new Tcl scripting property
2016.10.31	16.1.0	• Implemented Intel rebranding. • Implemented Qsys Pro rebranding. • Integrated Qsys Pro chapter with Qsys. • Added command-line options for qsys-archive. • Added command-line options for quartus_ipgenerate. • Updated the Qsys Pro scripting commands. • Added topic on Qsys Pro design conversion.
2016.05.03	16.0.0	• Qsys Pro Command-Line Utilities updated with latest supported command-line options. • Added: *Generate Header Files*
2015.11.02	15.1.0	• Added: *Troubleshooting IP or Qsys Pro System Upgrade*. • Added: *Generating Version-Agnostic IP and Qsys Pro Simulation Scripts*. • Changed instances of *Quartus II* to *Quartus Prime*.
2015.11.02	15.1.0	• Changed instances of *Quartus II* to *Quartus Prime*.
2015.05.04	15.0.0	• New figure: *Avalon-MM Write Master Timing Waveforms in the Parameters Tab*. • Added **Enable ECC protection** option, *Specify Qsys Pro Interconnect Requirements*. • Added External Memory Interface Debug Toolkit note, *Generate a Qsys Pro System*. • Modelsim-Altera now supports native mixed-language (VHDL/Verilog/SystemVerilog) simulation, *Generating Files for Synthesis and Simulation*.
December 2014	14.1.0	• Create and Manage Hierarchical Qsys Pro Systems. • Schematic tab. • View and Filter Clock and Reset Domains. • **File ➤ Recent Projects** menu item. • Updated example: Hierarchical System Using Instance Parameters

continued...

Date	Version	Changes
August 2014	14.0a10.0	• Added distinction between legacy and standard device generation. • Updated: *Upgrading Outdated IP Components*. • Updated: *Generating a Qsys Pro System*. • Updated: *Integrating a Qsys Pro System with the Quartus Prime Software*. • Added screen shot: *Displaying Your Qsys Pro System*.
June 2014	14.0.0	• Added tab descriptions: Details, Connections. • Added *Managing IP Settings in the Quartus Prime Software*. • Added *Upgrading Outdated IP Components*. • Added *Support for Avalon-MM Non-Power of Two Data Widths*.
November 2013	13.1.0	• Added *Integrating with the .qsys File*. • Added *Using the Hierarchy Tab*. • Added *Managing Interconnect Requirements*. • Added *Viewing Qsys Pro Interconnect*.
May 2013	13.0.0	• Added AMBA APB support. • Added qsys-generate utility. • Added VHDL BFM ID support. • Added *Creating Secure Systems (TrustZones)* . • Added *CMSIS Support for Qsys Pro Systems With An HPS Component*. • Added VHDL language support options.
November 2012	12.1.0	• Added AMBA AXI4 support.
June 2012	12.0.0	• Added AMBA AX3I support. • Added Preset Editor updates. • Added command-line utilities, and scripts.
November 2011	11.1.0	• Added Synopsys VCS and VCS MX Simulation Shell Script. • Added Cadence Incisive Enterprise (NCSIM) Simulation Shell Script. • Added *Using Instance Parameters and Example Hierarchical System Using Parameters*.
May 2011	11.0.0	• Added simulation support in Verilog HDL and VHDL. • Added testbench generation support. • Updated simulation and file generation sections.
December 2010	10.1.0	Initial release.

Related Links

Altera Documentation Archive

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

10 Creating Qsys Pro Components

You can create a Hardware Component Definition File (`_hw.tcl`) to describe and package IP components for use in a Qsys Pro system. A `_hw.tcl` describes IP components, interfaces and HDL files. Qsys Pro provides the Component Editor to help you create a simple `_hw.tcl` file.

The **Demo AXI Memory** example on the **Qsys Pro Design Examples** page of the Altera web site provides the full code examples that appear in the following topics.

Qsys Pro supports Avalon, AMBA AXI3 (version 1.0), AMBA AXI4 (version 2.0), AMBA AXI4-Lite (version 2.0), AMBA AXI4-Stream (version 1.0), and AMBA APB3 (version 1.0) interface specifications.

Related Links

- Avalon Interface Specifications
- AMBA Protocol Specifications
- Demo AXI Memory Example

10.1 Qsys Pro Components

A Qsys Pro component includes the following elements:

- Information about the component type, such as name, version, and author.
- HDL description of the component's hardware, including SystemVerilog, Verilog HDL, or VHDL files.
- A Synopsys* Design Constraints File `.sdc` that defines the component for synthesis and simulation.
- A `.ip` file that defines the component's parameters.
- A component's interfaces, including I/O signals.

10.1.1 Interface Support in Qsys Pro

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

Qsys Pro IP components can include the following interface types:

Table 113. IP Component Interface Types

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

10.1.2 Component Structure

Intel provides components automatically installed with the Quartus Prime software. You can obtain a list of Qsys Pro-compliant components provided by third-party IP developers on Altera's **Intellectual Property & Reference Designs** page by typing: **qsys certified** in the **Search** box, and then selecting **IP Core & Reference Designs**. Components are also provided with Intel development kits, which are listed on the **All Development Kits** page.

Every component is defined with a *<component_name>*`_hw.tcl file`, a text file written in the Tcl scripting language that describes the component to Qsys Pro. When you design your own custom component, you can create the `_hw.tcl` file manually, or by using the Qsys Pro Component Editor.

The Component Editor simplifies the process of creating `_hw.tcl` files by creating a file that you can edit outside of the Component Editor to add advanced procedures. When you edit a previously saved `_hw.tcl` file, Qsys Pro automatically backs up the earlier version as `_hw.tcl~`.

You can move component files into a new directory, such as a network location, so that other users can use the component in their systems. The `_hw.tcl` file contains relative paths to the other files, so if you move an `_hw.tcl` file, you should also move all the HDL and other files associated with it.

There are four component types:

- **Static**— static components always generate the same output, regardless of their parameterization. Components that instantiate static components must have only static children.
- **Generated**—generated component's fileset callback allows an instance of the component to create unique HDL design files based on the instance's parameter values.
- **Composed**—composed components are subsystems constructed from instances of other components. You can use a composition callback to manage the subsystem in a composed component.
- **Generic**—generic components allow instantiation of IP components without an HDL implementation. Generic components enable hierarchical isolation between system interconnect and IP components.

Related Links

- Create a Composed Component or Subsystem on page 612
- Add Component Instances to a Static or Generated Component on page 614

10.1.3 Component File Organization

A typical component uses the following directory structure where the names of the directories are not significant:

<component_directory>/

- *<hdl>/*—Contains the component HDL design files, for example `.v`, `.sv`, or `.vhd` files that contain the top-level module, along with any required constraint files.
- *<component_name>* `_hw.tcl`—The component description file.
- *<component_name>* `_sw.tcl`—The software driver configuration file. This file specifies the paths for the `.c` and `.h` files associated with the component, when required.
- *<software>/*—Contains software drivers or libraries related to the component.

Note: Refer to the *Nios II Software Developer's Handbook* for information about writing a device driver or software package suitable for use with the Nios II processor.

Related Links

Nios II Software Developer's Handbook
Refer to the "Nios II Software Build Tools" and "Overview of the Hardware Abstraction Layer" chapters.

10.1.4 Component Versions

Qsys Pro systems support multiple versions of the same component within the same system; you can create and maintain multiple versions of the same component.

If you have multiple `_hw.tcl` files for components with the same NAME module properties and different VERSION module properties, both versions of the component are available.

If multiple versions of the component are available in the IP Catalog, you can add a specific version of a component by right-clicking the component, and then selecting **Add version** <*version_number*>.

10.1.4.1 Upgrade IP Components to the Latest Version

When you open a Qsys Pro design, if Qsys Pro detects IP components that require regeneration, the **Upgrade IP Cores** dialog box appears and allows you to upgrade outdated components.

Components that you must upgrade in order to successfully compile your design appear in red. Status icons indicate whether a component is currently being regenerated, the component is encrypted, or that there is not enough information to determine the status of component. To upgrade a component, in the **Upgrade IP Cores** dialog box, select the component that you want to upgrade, and then click **Upgrade**. The Quartus Prime software maintains a list of all IP components associated with your design on the **Components** tab in the Project Navigator.

Related Links

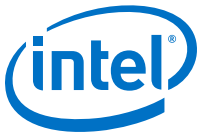
Upgrade IP Components Dialog Box

10.2 Design Phases of an IP Component

When you define a component with the Qsys Pro Component Editor, or a custom `_hw.tcl` file, you specify the information that Qsys Pro requires to instantiate the component in a Qsys Pro system and to generate the appropriate output files for synthesis and simulation.

The following phases describe the process when working with components in Qsys Pro:

- **Discovery**—During the discovery phase, Qsys Pro reads the `_hw.tcl` file to identify information that appears in the IP Catalog, such as the component's name, version, and documentation URLs. Each time you open Qsys Pro, the tool searches for the following file types using the default search locations and entries in the **IP Search Path**:
  - `_hw.tcl` files—Each `_hw.tcl` file defines a single component.
  - IP Index (`.ipx`) files—Each `.ipx` file indexes a collection of available components, or a reference to other directories to search.
- **Static Component Definition**—During the static component definition phase, Qsys Pro reads the `_hw.tcl` file to identify static parameter declarations, interface properties, interface signals, and HDL files that define the component. At this stage of the life cycle, the component interfaces may be only partially defined.
- **Parameterization**—During the parameterization phase, after an instance of the component is added to a Qsys Pro system, the user of the component specifies parameters with the component's parameter editor.
- **Validation**—During the validation phase, Qsys Pro validates the values of each instance's parameters against the allowed ranges specified for each parameter. You can use callback procedures that run during the validation phase to provide validation messages. For example, if there are dependencies between parameters where only certain combinations of values are supported, you can report errors for the unsupported values.



- **Elaboration**—During the elaboration phase, Qsys Pro queries the component for its interface information. Elaboration is triggered when an instance of a component is added to a system, when its parameters are changed, or when a system property changes. You can use callback procedures that run during the elaboration phase to dynamically control interfaces, signals, and HDL files based on the values of parameters. For example, interfaces defined with static declarations can be enabled or disabled during elaboration. When elaboration is complete, the component's interfaces and design logic must be completely defined.
- **Composition**—During the composition phase, a component can manipulate the instances in the component's subsystem. The **_hw.tcl** file uses a callback procedure to provide parameterization and connectivity of sub-components.
- **Generation**—During the generation phase, Qsys Pro generates synthesis or simulation files for each component in the system into the appropriate output directories, as well as any additional files that support associated tools

10.3 Create IP Components in the Qsys Pro Component Editor

The Qsys Pro Component Editor allows you to create and package an IP component. When you use the Component Editor to define a component, Qsys Pro writes the information to an `_hw.tcl` file.

The Qsys Pro Component Editor allows you to perform the following tasks:

- Specify component's identifying information, such as name, version, author, etc.
- Specify the SystemVerilog, Verilog HDL, VHDL files, and constraint files that define the component for synthesis and simulation.
- Create an HDL template to define a component interfaces, signals, and parameters.
- Set parameters on interfaces and signals that can alter the component's structure or functionality.

If you add the top-level HDL file that defines the component on **Files** tab in the Qsys Pro Component Editor, you must define the component's parameters and signals in the HDL file. You cannot add or remove them in the Component Editor.

If you do not have a top-level HDL component file, you can use the Qsys Pro Component Editor to add interfaces, signals, and parameters. In the Component Editor, the order in which the tabs appear reflects the recommended design flow for component development. You can use the **Prev** and **Next** buttons to guide you through the tabs.

In a Qsys Pro system, the interfaces of a component are connected in the system, or exported as top-level signals from the system.

If the component is not based on an existing HDL file, enter the parameters, signals, and interfaces first, and then return to the **Files** tab to create the top-level HDL file template. When you click **Finish**, Qsys Pro creates the component `_hw.tcl` file with the details that you enter in the Component Editor.

When you save the component, it appears in the IP Catalog.

If you require custom features that the Qsys Pro Component Editor does not support, for example, an elaboration callback, use the Component Editor to create the `_hw.tcl` file, and then manually edit the file to complete the component definition.

Note: If you add custom coding to a component, do not open the component file in the Qsys Pro Component Editor. The Qsys Pro Component Editor overwrites your custom edits.

Example 84. Qsys Pro Creates an _hw.tcl File from Entries in the Component Editor

```
#
# connection point clock
#
add_interface clock clock end
set_interface_property clock clockRate 0
set_interface_property clock ENABLED true

add_interface_port clock clk clk Input 1

#
# connection point reset
#
add_interface reset reset end
set_interface_property reset associatedClock clock
set_interface_property reset synchronousEdges DEASSERT
set_interface_property reset ENABLED true

add_interface_port reset reset_n reset_n Input 1

#
# connection point streaming
#
add_interface streaming avalon_streaming start
set_interface_property streaming associatedClock clock
set_interface_property streaming associatedReset reset
set_interface_property streaming dataBitsPerSymbol 8
set_interface_property streaming errorDescriptor ""
set_interface_property streaming firstSymbolInHighOrderBits true
set_interface_property streaming maxChannel 0
set_interface_property streaming readyLatency 0
set_interface_property streaming ENABLED true

add_interface_port streaming aso_data data Output 8
add_interface_port streaming aso_valid valid Output 1
add_interface_port streaming aso_ready ready Input 1

#
# connection point slave
#
add_interface slave axi end
set_interface_property slave associatedClock clock
set_interface_property slave associatedReset reset
set_interface_property slave readAcceptanceCapability 1
set_interface_property slave writeAcceptanceCapability 1
set_interface_property slave combinedAcceptanceCapability 1
set_interface_property slave readDataReorderingDepth 1
set_interface_property slave ENABLED true

add_interface_port slave axs_awid awid Input AXI_ID_W
...
add_interface_port slave axs_rresp rresp Output 2
```

Related Links

Component Interface Tcl Reference on page 762
Tcl commands allow you to perform a wide range of functions in Qsys Pro. Command descriptions contain the Qsys Pro phases where you can use the command, for example, main program, elaboration, composition, or fileset callback.

10.3.1 Save an IP Component and Create the _hw.tcl File

You save a component by clicking **Finish** in the Qsys Pro Component Editor. The Component Editor saves the component as <*component_name*> `_hw.tcl` file.

Intel recommends that you move `_hw.tcl` files and their associated files to an `ip/` directory within your Quartus Prime project directory. You can use IP components with other applications, such as the C compiler and a board support package (BSP) generator.

Refer to *Creating a System with Qsys Pro* for information on how to search for and add components to the IP Catalog for use in your designs.

Related Links

- Publishing Component Information to Embedded Software (Nios II Software Developer's Handbook)
- Creating a System with Qsys Pro on page 299
  Qsys Pro is a system integration tool included as part of the Quartus Prime software.

10.3.2 Edit an IP Component with the Qsys Pro Component Editor

In Qsys Pro, you make changes to a component by right-clicking the component in the **System Contents** tab, and then clicking **Edit**. After making changes, click **Finish** to save the changes to the `_hw.tcl` file.

You can open an `_hw.tcl` file in a text editor to view the hardware Tcl for the component. If you edit the `_hw.tcl` file to customize the component with advanced features, you cannot use the Component Editor to make further changes without over-writing your customized file.

You cannot use the Component Editor to edit components installed with the Quartus Prime software, such as Intel-provided components. If you edit the HDL for a component and change the interface to the top-level module, you must edit the component to reflect the changes you make to the HDL.

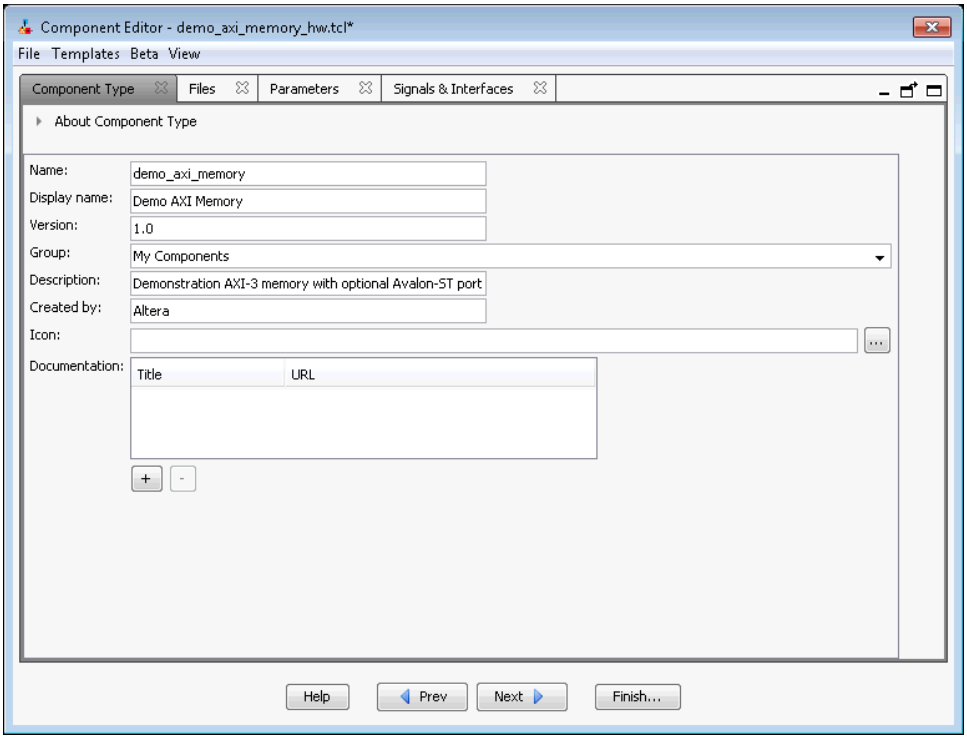
10.4 Specify IP Component Type Information

The **Component Type** tab in the Qsys Pro Component Editor allows you to specify the following information about the component:

- **Name**—Specifies the name used in the `_hw.tcl` filename, as well as in the top-level module name when you create a synthesis wrapper file for a non HDL-based component.
- **Display name**—Identifies the component in the parameter editor, which you use to configure and instance of the component, and also appears in the IP Catalog under **Project** and on the **System Contents** tab.
- **Version**—Specifies the version number of the component.
- **Group**—Represents the category of the component in the list of available components in the IP Catalog. You can select an existing group from the list, or define a new group by typing a name in the **Group** box. Separating entries in the **Group** box with a slash defines a subcategory. For example, if you type **Memories and Memory Controllers/On-Chip**, the component appears in the IP Catalog under the **On-Chip** group, which is a subcategory of the **Memories and Memory Controllers** group. If you save the component in the project directory, the component appears in the IP Catalog in the group you specified under **Project**. Alternatively, if you save the component in the Quartus Prime installation directory, the component appears in the specified group under **IP Catalog**.
- **Description**—Allows you to describe the component. This description appears when the user views the component details.
- **Created By**—Allows you to specify the author of the component.
- **Icon**—Allows you to enter the relative path to an icon file (`.gif`, `.jpg`, or `.png` format) that represents the component and appears as the header in the parameter editor for the component. The default image is the Altera MegaCore function icon.
- **Documentation**—Allows you to add links to documentation for the component, and appears when you right-click the component in the IP Catalog, and then select **Details**.
  - To specify an Internet file, begin your path with `http://`, for example: `http://mydomain.com/datasheets/my_memory_controller.html`.
  - To specify a file in the file system, begin your path with `file:///` for Linux, and `file:////` for Windows; for example (Windows): `file:////company_server/datasheets my_memory_controller.pdf`.

Figure 159. Component Type Tab in the Component Editor

The **Display name**, **Group**, **Description**, **Created By**, **Icon**, and **Documentation** entries are optional.



When you use the Component Editor to create a component, it writes this basic component information in the `_hw.tcl file`. The `package require` command specifies the Quartus Prime software version that Qsys Pro uses to create the `_hw.tcl` file, and ensures compatibility with this version of the Qsys Pro API in future ACDS releases.

Example 85. _hw.tcl Created from Entries in the Component Type Tab

The component defines its basic information with various module properties using the `set_module_property` command. For example, `set_module_property NAME` specifies the name of the component, while `set_module_property VERSION` allows you to specify the version of the component. When you apply a version to the `_hw.tcl` file, it allows the file to behave exactly the same way in future releases of the Quartus Prime software.

```
# request TCL package from ACDS 14.0

package require -exact qsys 14.0

# demo_axi_memory

set_module_property DESCRIPTION \
"Demo AXI-3 memory with optional Avalon-ST port"

set_module_property NAME demo_axi_memory
set_module_property VERSION 1.0
```

```
set_module_property GROUP "My Components"
set_module_property AUTHOR Altera
set_module_property DISPLAY_NAME "Demo AXI Memory"
```

Related Links

Component Interface Tcl Reference on page 762
Tcl commands allow you to perform a wide range of functions in Qsys Pro. Command descriptions contain the Qsys Pro phases where you can use the command, for example, main program, elaboration, composition, or fileset callback.

10.5 Create an HDL File in the Qsys Pro Component Editor

If you do not have an HDL file for your component, you can use the Qsys Pro Component Editor to define the component signals, interfaces, and parameters of your component, and then create a simple top-level HDL file.

You can then edit the HDL file to add the logic that describes the component's behavior.

1. In the Qsys Pro Component Editor, specify the information about the component in the **Signals & Interfaces**, and **Interfaces**, and **Parameters** tabs.
2. Click the **Files** tab.
3. Click **Create Synthesis File from Signals**.
   The Component Editor creates an HDL file from the specified signals, interfaces, and parameters, and the .v file appears in the **Synthesis File** table.

Related Links

Specify Synthesis and Simulation Files in the Qsys Pro Component Editor on page 596
The **Files** tab in the Qsys Pro Component Editor allows you to specify synthesis and simulation files for your custom component.

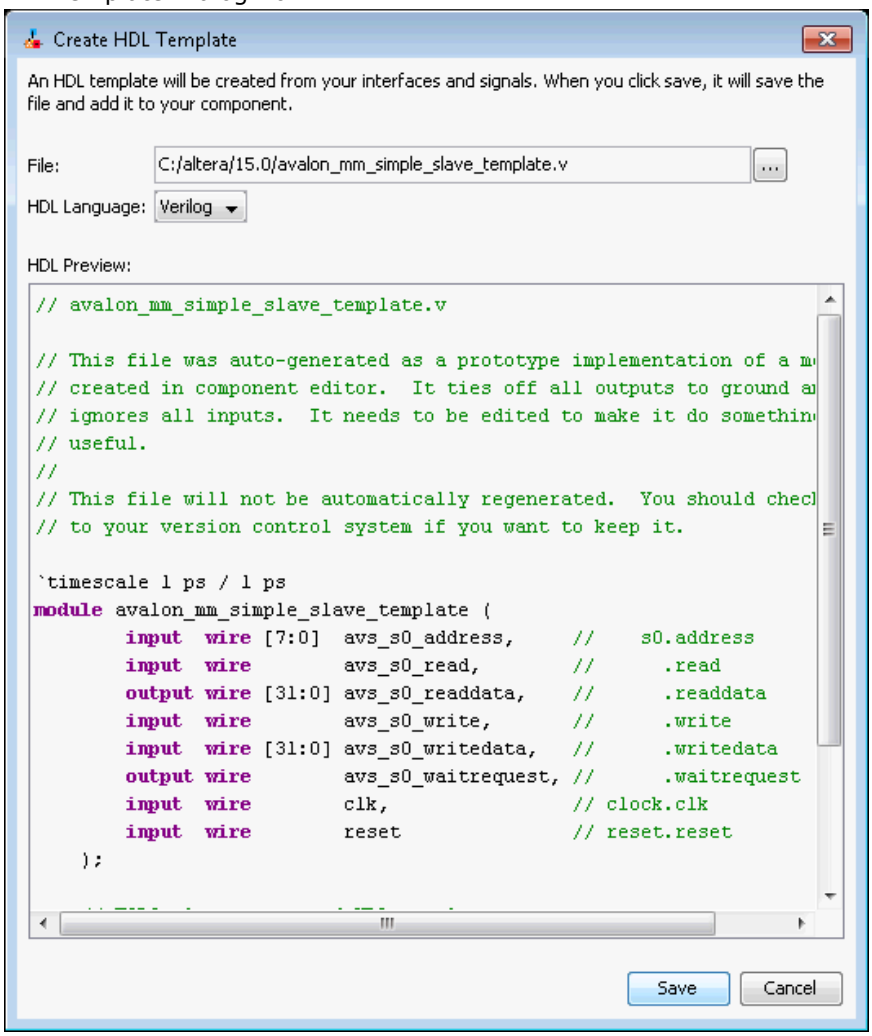
10.6 Create an HDL File Using a Template in the Qsys Pro Component Editor

You can use a template to create interfaces and signals for your Qsys Pro component

1. In Qsys Pro, click **New Component** in the IP Catalog.
2. On the **Component Type** tab, define your component information in the **Name**, **Display Name**, **Version**, **Group**, **Description**, **Created by**, **Icon**, and **Documentation** boxes.
3. Click **Finish**.
   Your new component appears in the IP Catalog under the category that you define for "Group".
4. In Qsys Pro, right-click your new component in the IP Catalog, and then click **Edit**.
5. In the Qsys Pro Component Editor, click any interface from the Templates drop-down menu.
   The Component Editor fills the **Signals** and **Interfaces** tabs with the component interface template details.

6. On the **Files** tab, click **Create Synthesis File from Signals**.
7. Do the following in the **Create HDL Template** dialog box as shown below:
   a. Verify that the correct files appears in **File** path, or browse to the location where you want to save your file.
   b. Select the HDL language.
   c. Click **Save** to save your new interface, or **Cancel** to discard the new interface definition.

Create HDL Template Dialog Box



8. Verify the ***<component_name>*.v** file appears in the **Synthesis Files** table on the **Files** tab.

Related Links

Specify Synthesis and Simulation Files in the Qsys Pro Component Editor on page 596

The **Files** tab in the Qsys Pro Component Editor allows you to specify synthesis and simulation files for your custom component.

10.7 Specify Synthesis and Simulation Files in the Qsys Pro Component Editor

The **Files** tab in the Qsys Pro Component Editor allows you to specify synthesis and simulation files for your custom component.

If you already an HDL files that describe the behavior and structure of your component, you can specify those files on the **Files** tab.

If you do not yet have an HDL file, you can specify the signals, interfaces, and parameters of the component in the Component Editor, and then use the **Create Synthesis File from Signals** option on the **Files** tab to create the top-level HDL file. The Component Editor generates the `_hw.tcl` commands to specify the files.

Note: After you analyze the component's top-level HDL file (on the **Files** tab), you cannot add or remove signals or change the signal names on the **Signals & Interfaces** tab. If you need to edit signals, edit your HDL source, and then click **Create Synthesis File from Signals** on the **Files** tab to integrate your changes.

A component uses filesets to specify the different sets of files that you can generate for an instance of the component. The supported fileset types are: `QUARTUS_SYNTH`, for synthesis and compilation in the Quartus Prime software, `SIM_VERILOG`, for Verilog HDL simulation, and `SIM_VHDL`, for VHDL simulation.

In an `_hw.tcl` file, you can add a fileset with the `add_fileset` command. You can then list specific files with the `add_fileset_file` command. The `add_fileset_property` command allows you to add properties such as `TOP_LEVEL`.

You can populate a fileset with a a fixed list of files, add different files based on a parameter value, or even generate an HDL file with a custom HDL generator function outside of the `_hw.tcl` file.

Related Links

- Create an HDL File in the Qsys Pro Component Editor on page 594
  If you do not have an HDL file for your component, you can use the Qsys Pro Component Editor to define the component signals, interfaces, and parameters of your component, and then create a simple top-level HDL file.
- Create an HDL File Using a Template in the Qsys Pro Component Editor on page 594
  You can use a template to create interfaces and signals for your Qsys Pro component

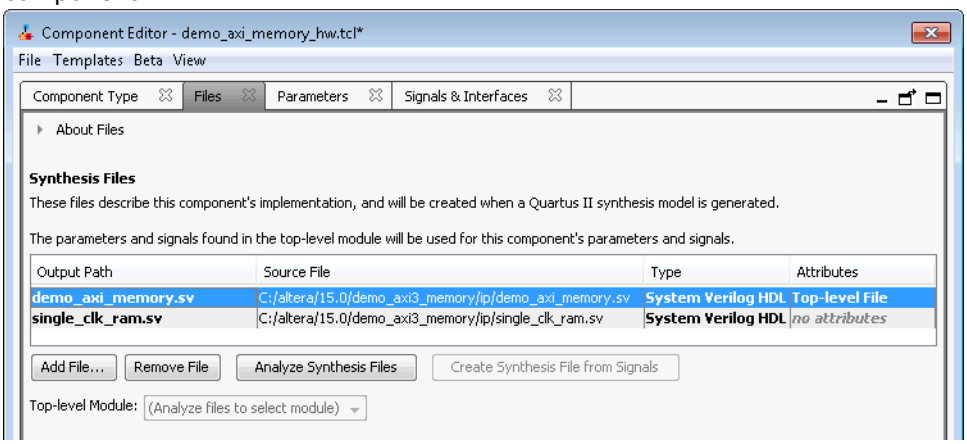
10.7.1 Specify HDL Files for Synthesis in the Qsys Pro Component Editor

In the Qsys Pro Component Editor, you can add HDL files and other support files with options on the `Files` tab.

A component must specify an HDL file as the top-level file. The top-level HDL file contains the top-level module. The **Synthesis Files** list may also include supporting HDL files, such as timing constraints, or other files required to successfully synthesize and compile in the Quartus Prime software. The synthesis files for a component are copied to the generation output directory during Qsys Pro system generation.

Figure 160. Using HDL Files to Define a Component

In the **Synthesis Files** section on the **Files** tab in the Qsys Pro Component Editor, the **demo_axi_memory.sv** file should be selected as the top-level file for the component.



10.7.2 Analyze Synthesis Files in the Qsys Pro Component Editor

After you specify the top-level HDL file in the Qsys Pro Component Editor, click **Analyze Synthesis Files** to analyze the parameters and signals in the top-level, and then select the top-level module from the **Top Level Module** list. If there is a single module or entity in the HDL file, Qsys Pro automatically populates the **Top-level Module** list.

Once analysis is complete and the top-level module is selected, you can view the parameters and signals on the **Parameters** and **Signals & Interfaces** tabs. The Component Editor may report errors or warnings at this stage, because the signals and interfaces are not yet fully defined.

Note: At this stage in the Component Editor flow, you cannot add or remove parameters or signals created from a specified HDL file without editing the HDL file itself.

The synthesis files are added to a fileset with the name QUARTUS_SYNTH and type QUARTUS_SYNTH in the `_hw.tcl` file created by the Component Editor. The top-level module is used to specify the TOP_LEVEL fileset property. Each synthesis file is individually added to the fileset. If the source files are saved in a different directory from the working directory where the `_hw.tcl` is located, you can use standard fixed or relative path notation to identify the file location for the PATH variable.

Example 86. _hw.tcl Created from Entries in the Files tab in the Synthesis Files Section

```
# file sets

add_fileset QUARTUS_SYNTH QUARTUS_SYNTH "" ""
set_fileset_property QUARTUS_SYNTH TOP_LEVEL demo_axi_memory

add_fileset_file demo_axi_memory.sv
SYSTEM_VERILOG PATH demo_axi_memory.sv

add_fileset_file single_clk_ram.v VERILOG PATH single_clk_ram.v
```

Related Links

- Specify HDL Files for Synthesis in the Qsys Pro Component Editor on page 596
  In the Qsys Pro Component Editor, you can add HDL files and other support files with options on the `Files` tab.
- Component Interface Tcl Reference on page 762
  Tcl commands allow you to perform a wide range of functions in Qsys Pro. Command descriptions contain the Qsys Pro phases where you can use the command, for example, main program, elaboration, composition, or fileset callback.

10.7.3 Name HDL Signals for Automatic Interface and Type Recognition in the Qsys Pro Component Editor

If you create the component's top-level HDL file before using the Component Editor, the Component Editor recognizes the interface and signal types based on the signal names in the source HDL file. This auto-recognition feature eliminates the task of manually assigning each interface and signal type in the Component Editor.

To enable auto-recognition, you must create signal names using the following naming convention:

<interface type prefix>_<interface name>_<signal type>

Specifying an interface name with *<interface name>* is optional if you have only one interface of each type in the component definition. For interfaces with only one signal, such as clock and reset inputs, the *<interface type prefix>* is also optional.

Table 114. Interface Type Prefixes for Automatic Signal Recognition

When the Component Editor recognizes a valid prefix and signal type for a signal, it automatically assigns an interface and signal type to the signal based on the naming convention. If no interface name is specified for a signal, you can choose an interface name on the **Signals & Interfaces** tab in the Component Editor.

Interface Prefix	Interface Type
asi	Avalon-ST sink (input)
aso	Avalon-ST source (output)
avm	Avalon-MM master
avs	Avalon-MM slave
axm	AXI master
axs	AXI slave
apm	APB master
aps	APB slave
coe	Conduit
csi	Clock Sink (input)
cso	Clock Source (output)
inr	Interrupt receiver
ins	Interrupt sender
ncm	Nios II custom instruction master

continued...

Interface Prefix	Interface Type
ncs	Nios II custom instruction slave
rsi	Reset sink (input)
rso	Reset source (output)
tcm	Avalon-TC master
tcs	Avalon-TC slave

Refer to the *Avalon Interface Specifications* or the *AMBA Protocol Specification* for the signal types available for each interface type.

Related Links

- Avalon Interface Specifications
- AMBA Protocol Specification

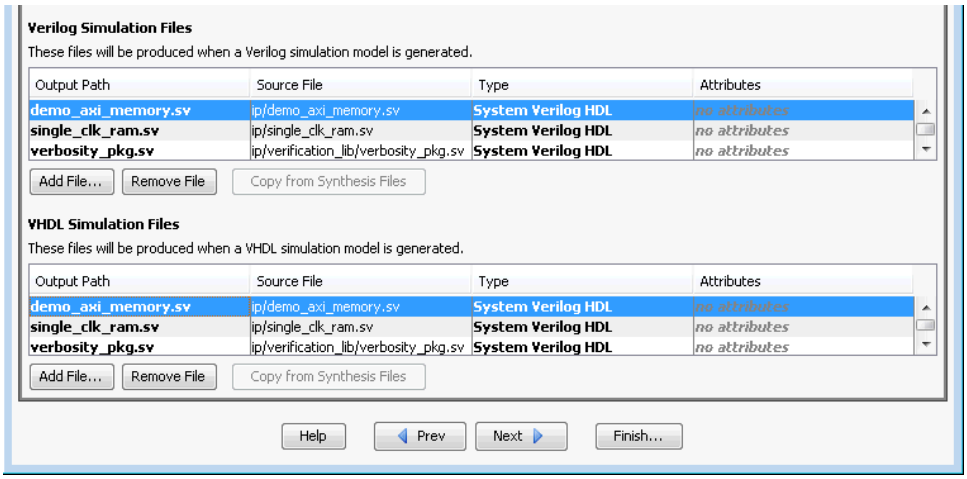
10.7.4 Specify Files for Simulation in the Component Editor

To support Qsys Pro system generation for your custom component, you must specify VHDL or Verilog simulation files.

You can choose to generate Verilog or VHDL simulation files. In most cases, these files are the same as the synthesis files. If there are simulation-specific HDL files or simulation models, you can use them in addition to, or in place of the synthesis files. To use your synthesis files as your simulation files, click **Copy From Synthesis Files** on the **Files** tab in the Qsys Pro Component Editor.

Note: The order that you add files to the fileset determines the order of compilation. For VHDL filesets with VHDL files, you must add the files bottom-up, adding the top-level file last.

Figure 161. Specifying the Simulation Output Files on the Files Tab



You specify the simulation files in a similar way as the synthesis files with the fileset commands in a `_hw.tcl` file. The code example below shows `SIM_VERILOG` and `SIM_VHDL` filesets for Verilog and VHDL simulation output files. In this example, the same Verilog files are used for both Verilog and VHDL outputs, and there is one

additional SystemVerilog file added. This method works for designers of Verilog IP to support users who want to generate a VHDL top-level simulation file when they have a mixed-language simulation tool and license that can read the Verilog output for the component.

Example 87. _hw.tcl Created from Entries in the Files tab in the Simulation Files Section

```
add_fileset SIM_VERILOG SIM_VERILOG "" ""
set_fileset_property SIM_VERILOG TOP_LEVEL demo_axi_memory
add_fileset_file single_clk_ram.v VERILOG PATH single_clk_ram.v

add_fileset_file verbosity_pkg.sv SYSTEM_VERILOG PATH \
verification_lib/verbosity_pkg.sv

add_fileset_file demo_axi_memory.sv SYSTEM_VERILOG PATH \
demo_axi_memory.sv

add_fileset SIM_VHDL SIM_VHDL "" ""
set_fileset_property SIM_VHDL TOP_LEVEL demo_axi_memory
set_fileset_property SIM_VHDL ENABLE_RELATIVE_INCLUDE_PATHS false

add_fileset_file demo_axi_memory.sv SYSTEM_VERILOG PATH \
demo_axi_memory.sv

add_fileset_file single_clk_ram.v VERILOG PATH single_clk_ram.v

add_fileset_file verbosity_pkg.sv SYSTEM_VERILOG PATH \
verification_lib/verbosity_pkg.sv
```

Related Links

Component Interface Tcl Reference on page 762
Tcl commands allow you to perform a wide range of functions in Qsys Pro. Command descriptions contain the Qsys Pro phases where you can use the command, for example, main program, elaboration, composition, or fileset callback.

10.7.5 Include an Internal Register Map Description in the .svd for Slave Interfaces Connected to an HPS Component

Qsys Pro supports the ability for IP component designers to specify register map information on their slave interfaces. This allows components with slave interfaces that are connected to an HPS component to include their internal register description in the generated `.svd` file.

To specify their internal register map, the IP component designer must write and generate their own `.svd` file and attach it to the slave interface using the following command:

`set_interface_property` <*slave interface*> `CMSIS_SVD_FILE` <*file path*>

The `CMSIS_SVD_VARIABLES` interface property allows for variable substitution inside the `.svd` file. You can dynamically modify the character data of the `.svd` file by using the CMSIS_SVD_VARIABLES property.

Example 88. Setting the CMSIS_SVD_VARIBLES Interface Property

For example, if you set the CMSIS_SVD_VARIABLES in the _hw tcl file, then in the .svd file if there is a variable {width} that describes the element <size>${width}</size>, it is replaced by <size>23</size> during generation of the .svd file. Note that substitution works only within character data (the data enclosed by <element>...</element>) and not on element attributes.

```
set_interface_property <interface name> \
CMSIS_SVD_VARIABLES "{width} {23}"
```

Related Links

- Component Interface Tcl Reference on page 762
  Tcl commands allow you to perform a wide range of functions in Qsys Pro. Command descriptions contain the Qsys Pro phases where you can use the command, for example, main program, elaboration, composition, or fileset callback.
- CMSIS - Cortex Microcontroller Software

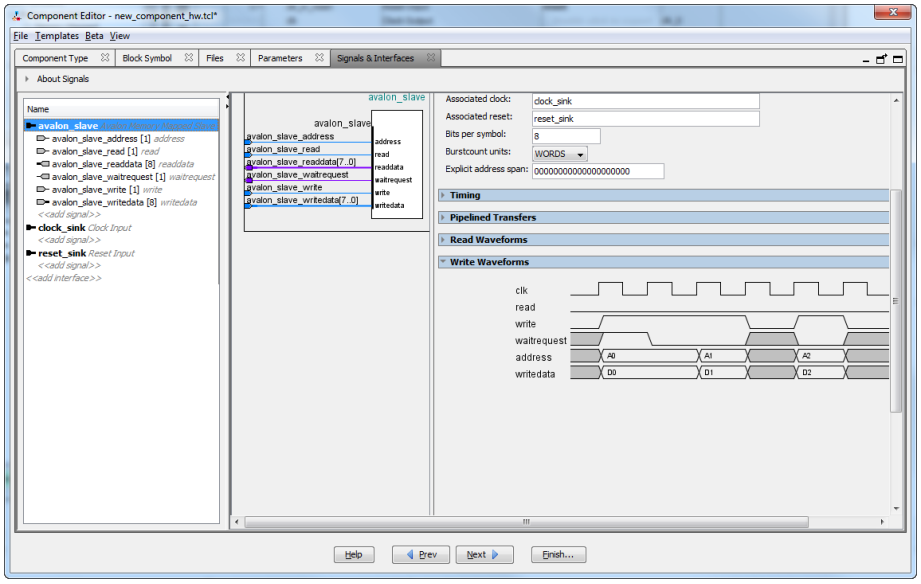
10.8 Add Signals and Interfaces in the Qsys Pro Component Editor

In the Qsys Pro Component Editor, the **Signals & Interfaces** tab allows you to add signals and interfaces for your custom IP component.

As you select interfaces and associated signals, you can customize the parameters. Messages appear as you add interfaces and signals to guide you when customizing the component. In the parameter editor, a block diagram displays for each interface. Some interfaces display waveforms to show the timing of the interface. If you update timing parameters, the waveforms update automatically.

1. In Qsys Pro, click ***New Component*** in the IP Catalog.
2. In the Qsys Pro Component Editor, click the **Signals & Interfaces** tab.
3. To add an interface, click <<*add interface*>> in the left pane.
   A drop-down list appears where you select the interface type.
4. Select an interface from the drop-down list.
   The selected interface appears in the parameter editor where you can specify its parameters.
5. To add signals for the selected interface click <<*add signal*>> below the selected interface.
6. To move signals between interfaces, select the signal, and then drag it to another interface.
7. To rename a nsignal or interface, select the element, and then press **F2**.
8. To remove a signal or interface, right-click the element, and then click **Remove**. Alternatively, to remove an signal or interface, you can select the element, and then press **Delete**. When you remove an interface, Qsys Pro also removes all of its associated signals.

Figure 162. Qsys Pro Signals & Interfaces tab



10.9 Specify Parameters in the Qsys Pro Component Editor

Components can include parameterized HDL, which allow users of the component flexibility in meeting their system requirements. For example, a component may have a configurable memory size or data width, where one HDL implementation can be used in different systems, each with unique parameters values.

The **Parameters** tab allows you specify the parameters that are used to configure instances of the component in a Qsys Pro system. You can specify various properties for each parameter that describe how to display and use the parameter. You can also specify a range of allowed values that are checked during the validation phase. The **Parameters** table displays the HDL parameters that are declared in the top-level HDL module. If you have not yet created the top-level HDL file, the parameters that you create on the **Parameters** tab are included in the top-level synthesis file template created from the **Files** tab.

When the component includes HDL files, the parameters match those defined in the top-level module, and you cannot be add or remove them on the **Parameters** tab. To add or remove the parameters, edit your HDL source, and then re-analyze the file.

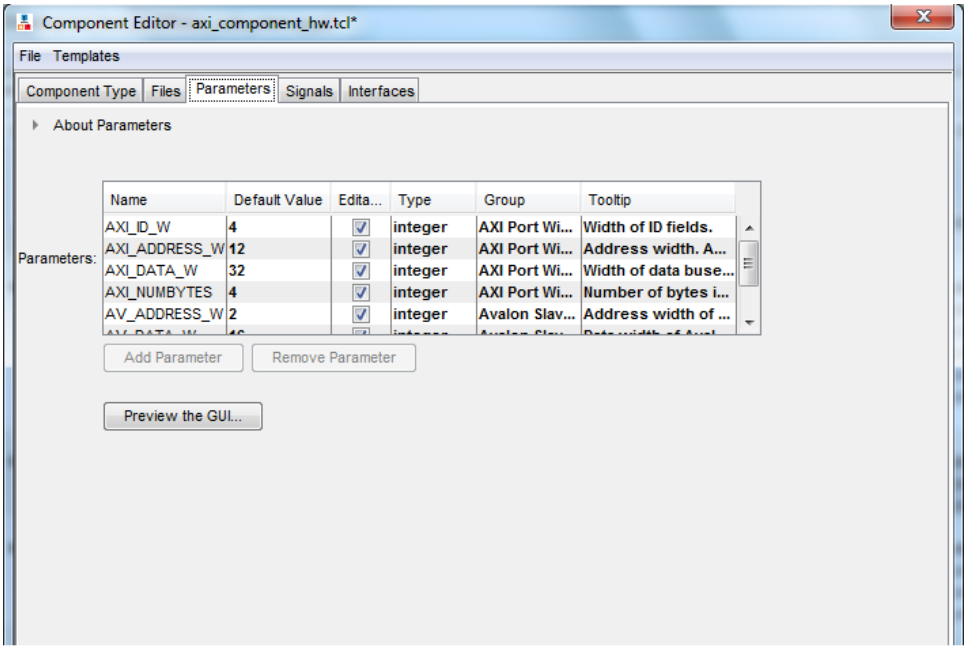
If you used the Component Editor to create a top-level template HDL file for synthesis, you can remove the newly-created file from the **Synthesis Files** list on the **Files** tab, make your parameter changes, and then re-analyze the top-level synthesis file.

You can use the **Parameters** table to specify the following information about each parameter:

- **Name**—Specifies the name of the parameter.
- **Default Value**—Sets the default value used in new instances of the component.
- **Editable**—Specifies whether or not the user can edit the parameter value.
- **Type**—Defines the parameter type as string, integer, boolean, std_logic, logic vector, natural, or positive.
- **Group**—Allows you to group parameters in parameter editor.
- **Tooltip**—Allows you to add a description of the parameter that appears when the user of the component points to the parameter in the parameter editor.

Figure 163. Parameters Tab in the Qsys Pro Components Editor

On the **Parameters** tab, you can click **Preview the GUI** at any time to see how the declared parameters appear in the parameter editor. Parameters with their default values appear with checks in the **Editable** column, indicating that users of this component are allowed to modify the parameter value. Editable parameters cannot contain computed expressions. You can group parameters under a common heading or section in the parameter editor with the **Group** column, and a tooltip helps users of the component understand the function of the parameter. Various parameter properties allow you to customize the component's parameter editor, such as using radio buttons for parameter selections, or displaying an image.



Example 89. _hw.tcl Created from Entries in the Parameters Tab

In this example, the first `add_parameter` command includes commonly-specified properties. The `set_parameter_property` command specifies each property individually. The **Tooltip** column on the **Parameters** tab maps to the `DESCRIPTION` property, and there is an additional unused `UNITS` property created in the code. The `HDL_PARAMETER` property specifies that the value of the parameter is specified in the

HDL instance wrapper when creating instances of the component. The **Group** column in the **Parameters** tab maps to the display items section with the `add_display_item` commands.

Note: If a parameter <*n*> defines the width of a signal, the signal width must follow the format: <*n*-1>:0.

```
#
# parameters
#
add_parameter AXI_ID_W INTEGER 4 "Width of ID fields"
set_parameter_property AXI_ID_W DEFAULT_VALUE 4
set_parameter_property AXI_ID_W DISPLAY_NAME AXI_ID_W
set_parameter_property AXI_ID_W TYPE INTEGER
set_parameter_property AXI_ID_W UNITS None
set_parameter_property AXI_ID_W DESCRIPTION "Width of ID fields"
set_parameter_property AXI_ID_W HDL_PARAMETER true
add_parameter AXI_ADDRESS_W INTEGER 12
set_parameter_property AXI_ADDRESS_W DEFAULT_VALUE 12

add_parameter AXI_DATA_W INTEGER 32
...
#
# display items
#
add_display_item "AXI Port Widths" AXI_ID_W PARAMETER ""
```

Note: If an AXI slave's ID bit width is smaller than required for your system, the AXI slave response may not reach all AXI masters. The formula of an AXI slave ID bit width is calculated as follows:

`maximum_master_id_width_in_the_interconnect + log2` (number_of_masters_in_the_same_interconnect)

For example, if an AXI slave connects to three AXI masters and the maximum AXI master ID length of the three masters is 5 bits, then the AXI slave ID is 7 bits, and is calculated as follows:

$5 \text{ bits} + 2 \text{ bits } (\log_2(3 \text{ masters})) = 7$

Table 115. AXI Master and Slave Parameters

Qsys Pro refers to AXI interface parameters to build AXI interconnect. If these parameter settings are incompatible with the component's HDL behavior, Qsys Pro interconnect and transactions may not work correctly. To prevent unexpected interconnect behavior, you must set the AXI component parameters.

AXI Master Parameters	AXI Slave Parameters
`readIssuingCapability`	`readAcceptanceCapability`
`writeIssuingCapability`	`writeAcceptanceCapability`
`combinedIssuingCapability`	`combinedAcceptanceCapability`
	`readDataReorderingDepth`

Related Links

Component Interface Tcl Reference on page 762
Tcl commands allow you to perform a wide range of functions in Qsys Pro. Command descriptions contain the Qsys Pro phases where you can use the

command, for example, main program, elaboration, composition, or fileset callback.

10.9.1 Valid Ranges for Parameters in the _hw.tcl File

In the `_hw.tcl` file, you can specify valid ranges for parameters.

Qsys Pro validation checks each parameter value against the ALLOWED_RANGES property. If the values specified are outside of the allowed ranges, Qsys Pro displays an error message. Specifying choices for the allowed values enables users of the component to choose the parameter value from a drop-down list or radio button in the parameter editor GUI instead of entering a value.

The ALLOWED_RANGES property is a list of valid ranges, where each range is a single value, or a range of values defined by a start and end value.

Table 116. ALLOWED_RANGES Property

ALLOWED_RANGES Property	Values
`{a b c}`	a, b, or c
`{"No Control" "Single Control" "Dual Controls"}`	Unique string values. Quotation marks are required if the strings include spaces .
`{1 2 4 8 16}`	1, 2, 4, 8, or 16
`{1:3}`	1 through 3, inclusive.
`{1 2 3 7:10}`	1, 2, 3, or 7 through 10 inclusive.

Related Links

Declare Parameters with Custom _hw.tcl Commands on page 607

10.9.2 Types of Qsys Pro Parameters

Qsys Pro uses the following parameter types: user parameters, system information parameters, and derived parameters.

Qsys Pro User Parameters on page 606

Qsys Pro System Information Parameters on page 606

Qsys Pro Derived Parameters on page 607

Related Links

Declare Parameters with Custom _hw.tcl Commands on page 607

10.9.2.1 Qsys Pro User Parameters

User parameters are parameters that users of a component can control, and appear in the parameter editor for instances of the component. User parameters map directly to parameters in the component HDL. For user parameter code examples, such as AXI_DATA_W and ENABLE_STREAM_OUTPUT, refer to *Declaring Parameters with Custom hw.tcl Commands*.

10.9.2.2 Qsys Pro System Information Parameters

A SYSTEM_INFO parameter is a parameter whose value is set automatically by the Qsys Pro system. When you define a SYSTEM_INFO parameter, you provide an `information type`, and additional arguments.

For example, you can configure a parameter to store the clock frequency driving a clock input for your component. To do this, define the parameter as SYSTEM_INFO of type CLOCK_RATE:

`set_parameter_property <param> SYSTEM_INFO CLOCK_RATE`

You then set the name of the clock interface as the SYSTEM_INFO argument:

`set_parameter_property <param> SYSTEM_INFO_ARG <clkname>`

10.9.2.2.1 Obtaining Device Trait Information Using PART_TRAIT System Information Parameter

Within Qsys Pro, an IP core can obtain information on the particular traits of a device using the PART_TRAIT system info parameter.
This system info parameter takes an argument corresponding to the desired part trait. The requested trait must match the trait name as specified in the device database.

Note: Using this API declares your core as dependent on the requested trait.

To get the part number setting of Qsys Pro system, use the value DEVICE, with the SYSTEM_INFO_ARG parameter property:

```
add_parameter part_trait_device string ""
set_parameter_property part_trait_device SYSTEM_INFO_TYPE PART_TRAIT
set_parameter_property part_trait_device SYSTEM_INFO_ARG DEVICE
```

To get the base device of the part number setting of Qsys Pro system, use the value BASE_DEVICE, with the SYSTEM_INFO_ARG parameter property:

```
add_parameter part_trait_bd string ""
set_parameter_property part_trait_bd SYSTEM_INFO_TYPE PART_TRAIT
set_parameter_property part_trait_bd SYSTEM_INFO_ARG BASE_DEVICE
```

To get the device speed-grade of the part number setting of Qsys Pro system, use the value DEVICE_SPEEDGRADE, with the SYSTEM_INFO_ARG parameter property:

```
add_parameter part_trait_sg string ""
set_parameter_property part_trait_sg SYSTEM_INFO_TYPE PART_TRAIT
set_parameter_property part_trait_sg SYSTEM_INFO_ARG DEVICE_SPEEDGRADE
```

10.9.2.3 Qsys Pro Derived Parameters

Derived parameter values are calculated from other parameters during the Elaboration phase, and are specified in the **hw.tcl** file with the DERIVED property. Derived parameter values are calculated from other parameters during the Elaboration phase, and are specified in the **hw.tcl** file with the DERIVED property. For example, you can derive a clock period parameter from a data rate parameter. Derived parameters are sometimes used to perform operations that are difficult to perform in HDL, such as using logarithmic functions to determine the number of address bits that a component requires.

Related Links

Declare Parameters with Custom _hw.tcl Commands on page 607

10.9.2.3.1 Parameterized Parameter Widths

Qsys Pro allows a `std_logic_vector` parameter to have a width that is defined by another parameter, similar to derived parameters. The width can be a constant or the name of another parameter.

10.9.3 Declare Parameters with Custom _hw.tcl Commands

The example below illustrates a custom `_hw.tcl` file, with more advanced parameter commands than those generated when you specify parameters in the Component Editor. Commands include the ALLOWED_RANGES property to provide a range of values for the AXI_ADDRESS_W (**Address Width**) parameter, and a list of parameter values for the AXI_DATA_W (**Data Width**) parameter. This example also shows the parameter AXI_NUMBYTES (**Data width in bytes**) parameter; that uses the DERIVED property. In addition, these commands illustrate the use of the GROUP property, which groups some parameters under a heading in the parameter editor GUI. You use the ENABLE_STREAM_OUTPUT_GROUP (**Include Avalon streaming source port**) parameter to enable or disable the optional Avalon-ST interface in this design, and is displayed as a check box in the parameter editor GUI because the parameter is of type BOOLEAN. Refer to figure below to see the parameter editor GUI resulting from these **hw.tcl** commands.

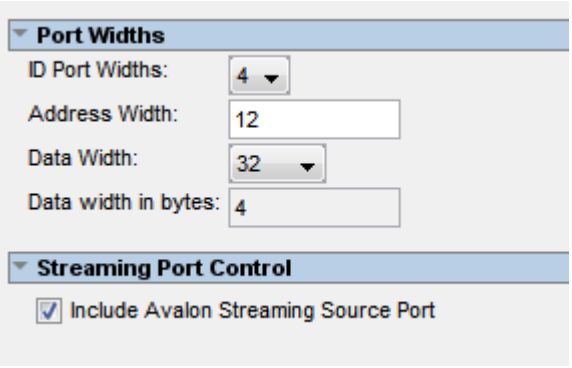
Example 90. Parameter Declaration

In this example, the AXI_NUMBYTES parameter is derived during the Elaboration phase based on another parameter, instead of being assigned to a specific value. AXI_NUMBYTES describes the number of bytes in a word of data. Qsys Pro calculates the AXI_NUMBYTES parameter from the DATA_WIDTH parameter by dividing by 8. The `_hw.tcl` code defines the AXI_NUMBYTES parameter as a derived parameter, since its value is calculated in an elaboration callback procedure. The AXI_NUMBYTES parameter value is not editable, because its value is based on another parameter value.

```
add_parameter AXI_ADDRESS_W INTEGER 12

set_parameter_property AXI_ADDRESS_W DISPLAY_NAME \
"AXI Slave Address Width"

set_parameter_property AXI_ADDRESS_W DESCRIPTION \
"Address width."

set_parameter_property AXI_ADDRESS_W UNITS bits
```

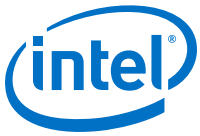
```
set_parameter_property AXI_ADDRESS_W ALLOWED_RANGES 4:16
set_parameter_property AXI_ADDRESS_W HDL_PARAMETER true

set_parameter_property AXI_ADDRESS_W GROUP \
"AXI Port Widths"

add_parameter AXI_DATA_W INTEGER 32
set_parameter_property AXI_DATA_W DISPLAY_NAME "Data Width"

set_parameter_property AXI_DATA_W DESCRIPTION \
"Width of data buses."

set_parameter_property AXI_DATA_W UNITS bits

set_parameter_property AXI_DATA_W ALLOWED_RANGES \
{8 16 32 64 128 256 512 1024}

set_parameter_property AXI_DATA_W HDL_PARAMETER true
set_parameter_property AXI_DATA_W GROUP "AXI Port Widths"

add_parameter AXI_NUMBYTES INTEGER 4
set_parameter_property AXI_NUMBYTES DERIVED true

set_parameter_property AXI_NUMBYTES DISPLAY_NAME \
"Data Width in bytes; Data Width/8"

set_parameter_property AXI_NUMBYTES DESCRIPTION \
"Number of bytes in one word"

set_parameter_property AXI_NUMBYTES UNITS bytes
set_parameter_property AXI_NUMBYTES HDL_PARAMETER true
set_parameter_property AXI_NUMBYTES GROUP "AXI Port Widths"

add_parameter ENABLE_STREAM_OUTPUT BOOLEAN true

set_parameter_property ENABLE_STREAM_OUTPUT DISPLAY_NAME \
"Include Avalon Streaming Source Port"

set_parameter_property ENABLE_STREAM_OUTPUT DESCRIPTION \
"Include optional Avalon-ST source (default),\
or hide the interface"

set_parameter_property ENABLE_STREAM_OUTPUT GROUP \
"Streaming Port Control"

...
```

Figure 164. Resulting Parameter Editor GUI from Parameter Declarations



Related Links

- Control Interfaces Dynamically with an Elaboration Callback on page 609



- Component Interface Tcl Reference on page 762
  Tcl commands allow you to perform a wide range of functions in Qsys Pro. Command descriptions contain the Qsys Pro phases where you can use the command, for example, main program, elaboration, composition, or fileset callback.

10.9.4 Validate Parameter Values with a Validation Callback

You can use a validation callback procedure to validate parameter values with more complex validation operations than the ALLOWED_RANGES property allows. You define a validation callback by setting the VALIDATION_CALLBACK module property to the name of the Tcl callback procedure that runs during the validation phase. In the validation callback procedure, the current parameter values is queried, and warnings or errors are reported about the component's configuration.

Example 91. Demo AXI Memory Example

If the optional Avalon streaming interface is enabled, then the control registers must be wide enough to hold an AXI RAM address, so the designer can add an error message to ensure that the user enters allowable parameter values.

```
set_module_property VALIDATION_CALLBACK validate
proc validate {} {
if {
  [get_parameter_value ENABLE_STREAM_OUTPUT ] &&
  ([get_parameter_value AXI_ADDRESS_W] >
  [get_parameter_value AV_DATA_W])
}
send_message error "If the optional Avalon streaming port\
is enabled, the AXI Data Width must be equal to or greater\
than the Avalon control port Address Width"
}
}
```

Related Links

- Component Interface Tcl Reference on page 762
  Tcl commands allow you to perform a wide range of functions in Qsys Pro. Command descriptions contain the Qsys Pro phases where you can use the command, for example, main program, elaboration, composition, or fileset callback.
- Demo AXI Memory Example

10.10 Control Interfaces Dynamically with an Elaboration Callback

You can allow user parameters to dynamically control your component's behavior with a an elaboration callback procedure during the elaboration phase. Using an elaboration callback allows you to change interface properties, remove interfaces, or add new interfaces as a function of a parameter value. You define an elaboration callback by setting the module property ELABORATION_CALLBACK to the name of the Tcl callback procedure that runs during the elaboration phase. In the callback procedure, you can query the parameter values of the component instance, and then change the interfaces accordingly.

Example 92. Avalon-ST Source Interface Optionally Included in a Component Specified with an Elaboration Callback

```
set_module_property ELABORATION_CALLBACK elaborate

proc elaborate {} {

   # Optionally disable the Avalon- ST data output

    if{[ get_parameter_value ENABLE_STREAM_OUTPUT] == "false" }{
       set_port_property aso_data    termination true
       set_port_property aso_valid   termination true
       set_port_property aso_ready   termination true
       set_port_property aso_ready   termination_value 0
    }
  # Calculate the Data Bus Width in bytes

    set bytewidth_var [expr [get_parameter_value AXI_DATA_W]/8]
    set_parameter_value AXI_NUMBYTES $bytewidth_var
}
```

Related Links

- Declare Parameters with Custom _hw.tcl Commands on page 607
- Validate Parameter Values with a Validation Callback on page 609
- Component Interface Tcl Reference on page 762
  Tcl commands allow you to perform a wide range of functions in Qsys Pro. Command descriptions contain the Qsys Pro phases where you can use the command, for example, main program, elaboration, composition, or fileset callback.

10.11 Control File Generation Dynamically with Parameters and a Fileset Callback

You can use a fileset callback to control which files are created in the output directories during the generation phase based on parameter values, instead of providing a fixed list of files. In a callback procedure, you can query the values of the parameters and use them to generate the appropriate files. To define a fileset callback, you specify a callback procedure name as an argument in the `add_fileset` command. You can use the same fileset callback procedure for all of the filesets, or create separate procedures for synthesis and simulation, or Verilog and VHDL.

Example 93. Fileset Callback Using Parameters to Control Filesets in Two Different Ways

The RAM_VERSION parameter chooses between two different source files to control the implementation of a RAM block. For the top-level source file, a custom Tcl routine generates HDL that optionally includes control and status registers, depending on the value of the CSR_ENABLED parameter.

During the generation phase, Qsys Pro creates a a top-level Qsys Pro system HDL wrapper module to instantiate the component top-level module, and applies the component's parameters, for any parameter whose parameter property HDL_PARAMETER is set to true.

```
#Create synthesis fileset with fileset_callback and set top level

add_fileset my_synthesis_fileset QUARTUS_SYNTH fileset_callback
```

```
set_fileset_property my_synthesis_fileset TOP_LEVEL \
demo_axi_memory

# Create Verilog simulation fileset with same fileset_callback
# and set top level

add_fileset my_verilog_sim_fileset SIM_VERILOG fileset_callback

set_fileset_property my_verilog_sim_fileset TOP_LEVEL \
demo_axi_memory

# Add extra file needed for simulation only

add_fileset_file verbosity_pkg.sv SYSTEM_VERILOG PATH \
verification_lib/verbosity_pkg.sv

# Create VHDL simulation fileset (with Verilog files
# for mixed-language VHDL simulation)

add_fileset my_vhdl_sim_fileset SIM_VHDL fileset_callback
set_fileset_property my_vhdl_sim_fileset TOP_LEVEL demo_axi_memory

add_fileset_file verbosity_pkg.sv SYSTEM_VERILOG PATH
verification_lib/verbosity_pkg.sv

# Define parameters required for fileset_callback

add_parameter RAM_VERSION INTEGER 1
set_parameter_property RAM_VERSION ALLOWED_RANGES {1 2}
set_parameter_property RAM_VERSION HDL_PARAMETER false
add_parameter CSR_ENABLED BOOLEAN enable
set_parameter_property CSR_ENABLED HDL_PARAMETER false

# Create Tcl callback procedure to add appropriate files to
# filesets based on parameters

proc fileset_callback { entityName } {
  send_message INFO "Generating top-level entity $entityName"
  set ram [get_parameter_value RAM_VERSION]
  set csr_enabled [get_parameter_value CSR_ENABLED]

  send_message INFO "Generating memory
  implementation based on RAM_VERSION $ram    "

      if {$ram == 1} {
          add_fileset_file single_clk_ram1.v VERILOG PATH \
    single_clk_ram1.v
      } else     {
          add_fileset_file single_clk_ram2.v VERILOG PATH \
    single_clk_ram2.v
    }

send_message INFO "Generating top-level file for \
CSR_ENABLED $csr_enabled"

generate_my_custom_hdl $csr_enabled demo_axi_memory_gen.sv

add_fileset_file demo_axi_memory_gen.sv VERILOG PATH \
demo_axi_memory_gen.sv
}
```

Related Links

- Specify Synthesis and Simulation Files in the Qsys Pro Component Editor on page 596
  The **Files** tab in the Qsys Pro Component Editor allows you to specify synthesis and simulation files for your custom component.

- Component Interface Tcl Reference on page 762
  Tcl commands allow you to perform a wide range of functions in Qsys Pro. Command descriptions contain the Qsys Pro phases where you can use the command, for example, main program, elaboration, composition, or fileset callback.

10.12 Create a Composed Component or Subsystem

A composed component is a subsystem containing instances of other components. Unlike an HDL-based component, a composed component's HDL is created by generating HDL for the components in the subsystem, in addition to the Qsys Pro interconnect to connect the subsystem instances.

You can add child instances in a composition callback of the `_hw.tcl` file.

With a composition callback, you can also instantiate and parameterize sub-components as a function of the composed component's parameter values. You define a composition callback by setting the `COMPOSITION_CALLBACK` module property to the name of the composition callback procedures.

A composition callback replaces the validation and elaboration phases. HDL for the subsystem is generated by generating all of the sub-components and the top-level that combines them.

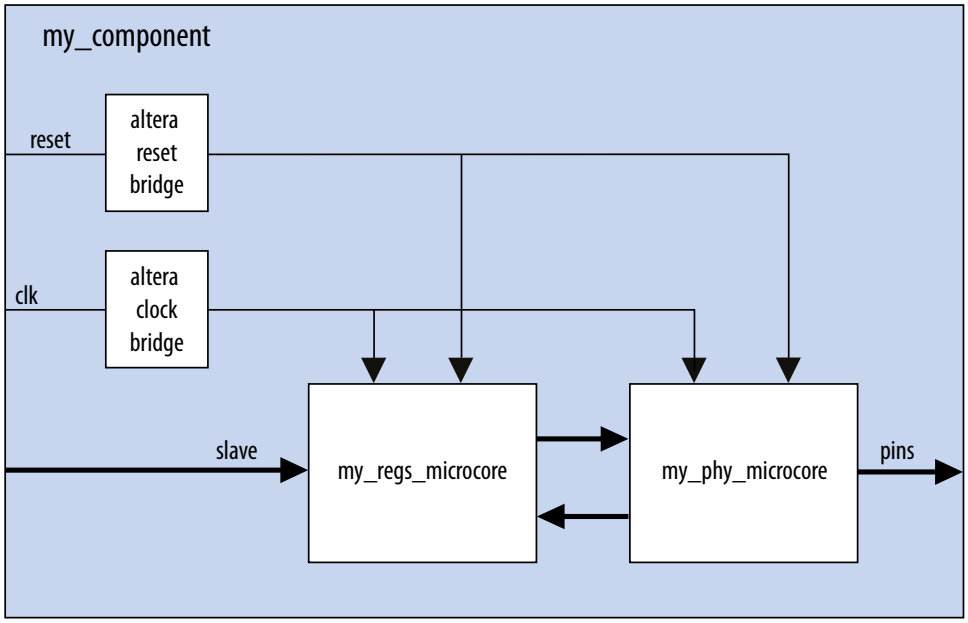
To connect instances of your component, you must define the component's interfaces. Unlike an HDL-based component, a composed component does not directly specify the signals that are exported. Instead, interfaces of submodules are chosen as the external interface, and each internal interface's ports are connected through the exported interface.

Exporting an interface means that you are making the interface visible from the outside of your component, instead of connecting it internally. You can set the `EXPORT_OF` property of the externally visible interface from the main program or the composition callback, to indicate that it is an exported view of the submodule's interface.

Exporting an interface is different than defining an interface. An exported interface is an exact copy of the subcomponent's interface, and you are not allowed to change properties on the exported interface. For example, if the internal interface is a 32-bit or 64-bit master without bursting, then the exported interface is the same. An interface on a subcomponent cannot be exported and also connected within the subsystem.

When you create an exported interface, the properties of the exported interface are copied from the subcomponent's interface without modification. Ports are copied from the subcomponent's interface with only one modification; the names of the exported ports on the composed component are chosen to ensure that they are unique.

Figure 165. Top-Level of a Composed Component



Example 94. Composed _hw.tcl File that Instantiates Two Sub-Components

Qsys Pro connects the components, and also connects the clocks and resets. Note that clock and reset bridge components are required to allow both sub-components to see common clock and reset inputs.

```
package require -exact qsys 14.0
set_module_property name my_component
set_module_property COMPOSITION_CALLBACK composed_component

proc composed_component {} {
  add_instance clk altera_clock_bridge
  add_instance reset altera_reset_bridge
  add_instance regs my_regs_microcore
  add_instance phy my_phy_microcore

  add_interface clk clock end
  add_interface reset reset end
  add_interface slave avalon slave
  add_interface pins conduit end

  set_interface_property clk EXPORT_OF clk.in_clk
  set_instance_property_value reset synchronous_edges deassert
  set_interface_property reset EXPORT_OF reset.in_reset
  set_interface_property slave EXPORT_OF regs.slave
  set_interface_property pins  EXPORT_OF phy.pins

  add_connection clk.out_clk reset.clk
  add_connection clk.out_clk regs.clk
  add_connection clk.out_clk phy.clk
  add_connection reset.out_reset regs.reset
  add_connection reset.out_reset phy.clk_reset
  add_connection regs.output phy.input
  add_connection phy.output regs.input
}
```

Related Links

Component Interface Tcl Reference on page 762
Tcl commands allow you to perform a wide range of functions in Qsys Pro. Command descriptions contain the Qsys Pro phases where you can use the command, for example, main program, elaboration, composition, or fileset callback.

10.13 Add Component Instances to a Static or Generated Component

You can create nested components by adding component instances to an existing component. Both static and generated components can create instances of other components. You can add child instances of a component in a `_hw.tcl` using elaboration callback.

With an elaboration callback, you can also instantiate and parameterize sub-components with the `add_hdl_instance` command as a function of the parent component's parameter values.

When you instantiate multiple nested components, you must create a unique variation name for each component with the `add_hdl_instance` command. Prefixing a variation name with the parent component name prevents conflicts in a system. The variation name can be the same across multiple parent components if the generated parameterization of the nested component is exactly the same.

Note: If you do not adhere to the above naming variation guidelines, Qsys Pro validation-time errors occur, which are often difficult to debug.

Related Links

- Static Components on page 614
  Static components always generate the same output, regardless of their parameterization. Components that instantiate static components must have only static children.
- Generated Components on page 616
  A generated component's fileset callback allows an instance of the component to create unique HDL design files based on the instance's parameter values. For example, you can write a fileset callback to include a control and status interface based on the value of a parameter. The callback overcomes a limitation of HDL languages, which do not allow run-time parameters.

10.13.1 Static Components

Static components always generate the same output, regardless of their parameterization. Components that instantiate static components must have only static children.

A design file that is static between all parameterizations of a component can only instantiate other static design files. Since static IPs always render the same HDL regardless of parameterization, Qsys Pro generates static IPs only once across multiple instantiations, meaning they have the same top-level name set.

Example 95. Typical Usage of the add_hdl_instance Command for Static Components

```
package require -exact qsys 14.0

set_module_property name add_hdl_instance_example
add_fileset synth_fileset QUARTUS_SYNTH synth_callback
set_fileset_property synth_fileset TOP_LEVEL basic_static
set_module_property elaboration_callback elab

proc elab {} {
  # Actual API to instantiate an IP Core
  add_hdl_instance emif_instance_name altera_mem_if_ddr3_emif

  # Make sure the parameters are set appropriately
  set_instance_parameter_value emif_instance_name SPEED_GRADE {7}
  ...
  }
proc synth_callback { output_name } {
  add_fileset_file "basic_static.v" VERILOG PATH basic_static.v
}
```

Example 96. Top-Level HDL Instance and Wrapper File Created by Qsys Pro

In this example, Qsys Pro generates a wrapper file for the instance name specified in the `_hw.tcl` file.

```
//Top Level Component HDL
module basic_static (input_wire, output_wire, inout_wire);
input [31:0] input_wire;
output [31:0] output_wire;
inout [31:0] inout_wire;

// Instantiation of the instance added via add_hdl_instance
// command. This is an example of how the instance added via
// the add_hdl_instance command can be used
// in the top-level file of the component.

emif_instance_name fixed_name_instantiation_in_top_level(
.pll_ref_clk (input_wire), // pll_ref_clk.clk
.global_reset_n (input_wire), // global_reset.reset_n
.soft_reset_n (input_wire), // soft_reset.reset_n
...
... );
endmodule

//Wrapper for added HDL instance
// emif_instance_name.v
// Generated using ACDS version 14.0

`timescale 1 ps / 1 ps
module emif_instance_name (
input wire pll_ref_clk, // pll_ref_clk.clk
input wire global_reset_n, // global_reset.reset_n
input wire soft_reset_n, // soft_reset.reset_n
output wire afi_clk, // afi_clk.clk
...
...);
example_addhdlinstance_system
_add_hdl_instance_example_0_emif_instance
_name_emif_instance_name emif_instance_name (

.pll_ref_clk (pll_ref_clk), // pll_ref_clk.clk
.global_reset_n (global_reset_n), // global_reset.reset_n
.soft_reset_n (soft_reset_n), // soft_reset.reset_n
```

```
...
...);
endmodule
```

10.13.2 Generated Components

A generated component's fileset callback allows an instance of the component to create unique HDL design files based on the instance's parameter values. For example, you can write a fileset callback to include a control and status interface based on the value of a parameter. The callback overcomes a limitation of HDL languages, which do not allow run-time parameters.

Generated components change their generation output (HDL) based on their parameterization. If a component is generated, then any component that may instantiate it with multiple parameter sets must also be considered generated, since its HDL changes with its parameterization. This case has an effect that propagates up to the top-level of a design.

Since generated components are generated for each unique parameterized instantiation, when implementing the `add_hdl_instance` command, you cannot use the same fixed name (specified using `instance_name`) for the different variants of the child HDL instances. To facilitate unique naming for the wrapper of each unique parameterized instantiation of child HDL instances, you must use the following command so that Qsys Pro generates a unique name for each wrapper. You can then access this unique wrapper name with a fileset callback so that the instances are instantiated inside the component's top-level HDL.

- To declare auto-generated fixed names for wrappers, use the command:

```
set_instance_property instance_name HDLINSTANCE_USE_GENERATED_NAME \
true
```

  Note: You can only use this command with a generated component in the global context, or in an elaboration callback.

- To obtain auto-generated fixed name with a fileset callback, use the command:

```
get_instance_property instance_name HDLINSTANCE_GET_GENERATED_NAME
```

  Note: You can only use this command with a fileset callback. This command returns the value of the auto-generated fixed name, which you can then use to instantiate inside the top-level HDL.

Example 97. Typical Usage of the add_hdl_instance Command for Generated Components

Qsys Pro generates a wrapper file for the instance name specified in the `_hw.tcl` file.

```
package require -exact qsys 14.0
set_module_property name generated_toplevel_component
set_module_property ELABORATION_CALLBACK elaborate
add_fileset QUARTUS_SYNTH QUARTUS_SYNTH generate
add_fileset SIM_VERILOG SIM_VERILOG generate
add_fileset SIM_VHDL SIM_VHDL generate

proc elaborate {} {

 # Actual API to instantiate an IP Core
 add_hdl_instance emif_instance_name altera_mem_if_ddr3_emif
```

```
 # Make sure the parameters are set appropriately
 set_instance_parameter_value emif_instance_name SPEED_GRADE {7}
 ...
 # instruct Qsys Pro to use auto generated fixed name
 set_instance_property emif_instance_name \
 HDLINSTANCE_USE_GENERATED_NAME 1
}

proc generate { entity_name } {

 # get the autogenerated name for emif_instance_name added
 # via add_hdl_instance

 set autogeneratedfixedname [get_instance_property \
 emif_instance_name HDLINSTANCE_GET_GENERATED_NAME]

 set fileID [open "generated_toplevel_component.v" r]
 set temp ""

 # read the contents of the file

 while {[eof $fileID] != 1} {
 gets $fileID lineInfo

 # replace the top level entity name with the name provided
 # during generation

 regsub -all "substitute_entity_name_here" $lineInfo \
 "${entity_name}" lineInfo

 # replace the autogenerated name for emif_instance_name added
 # via add_hdl_instance

 regsub -all "substitute_autogenerated_emifinstancename_here" \
 $lineInfo"${autogeneratedfixedname}" lineInfo \
 append temp "${lineInfo}\n"
}

# adding a top level component file

add_fileset_file ${entity_name}.v VERILOG TEXT $temp
}
```

Example 98. Top-Level HDL Instance and Wrapper File Created By Qsys Pro

```
// Top Level Component HDL

module substitute_entity_name_here (input_wire, output_wire,
inout_wire);

input [31:0] input_wire;
output [31:0] output_wire;
inout [31:0] inout_wire;

// Instantiation of the instance added via add_hdl_instance
// command. This is an example of how the instance added
// via add_hdl_instance command can be used
// in the top-level file of the component.

substitute_autogenerated_emifinstancename_here
fixed_name_instantiation_in_top_level (
.pll_ref_clk (input_wire), // pll_ref_clk.clk
.global_reset_n (input_wire), // global_reset.reset_n
.soft_reset_n (input_wire), // soft_reset.reset_n
...
... );
endmodule

// Wrapper for added HDL instance
```

```
// generated_toplevel_component_0_emif_instance_name.v is the
// auto generated //emif_instance_name
// Generated using ACDS version 13.

`timescale 1 ps / 1 ps
module generated_toplevel_component_0_emif_instance_name (
input wire pll_ref_clk, // pll_ref_clk.clk
input wire global_reset_n, // global_reset.reset_n
input wire soft_reset_n, // soft_reset.reset_n
output wire afi_clk, // afi_clk.clk
...
...);
example_addhdlinstance_system_add_hdl_instance_example_0_emif
_instance_name_emif_instance_name emif_instance_name (

.pll_ref_clk (pll_ref_clk), // pll_ref_clk.clk
.global_reset_n (global_reset_n), // global_reset.reset_n
.soft_reset_n (soft_reset_n), // soft_reset.reset_n
...
...);
endmodule
```

Related Links

- Intellectual Property & Reference Designs
- Control File Generation Dynamically with Parameters and a Fileset Callback on page 610

10.13.3 Design Guidelines for Adding Component Instances

In order to promote standard and predictable results when generating static and generated components, Intel recommends the following best-practices:

- For two different parameterizations of a component, a component must never generate a file of the same name with different instantiations. The contents of a file of the same name must be identical for every parameterization of the component.
- If a component generates a nested component, it must never instantiate two different parameterizations of the nested component using the same instance name. If the parent component's parameterization affects the parameters of the nested component, the parent component must use a unique instance name for each unique parameterization of the nested component.
- Static components that generate differently based on parameterization have the potential to cause problems in the following cases:
  - Different file names with the same entity names, results in same entity conflicts at compilation-time
  - Different contents with the same file name results in overwriting other instances of the component, and in either file, compile-time conflicts or unexpected behavior.
- Generated components that generate files not based on the output name and that have different content results in either compile-time conflicts, or unexpected behavior.

10.14 Adding a Generic Component to the Qsys Pro System

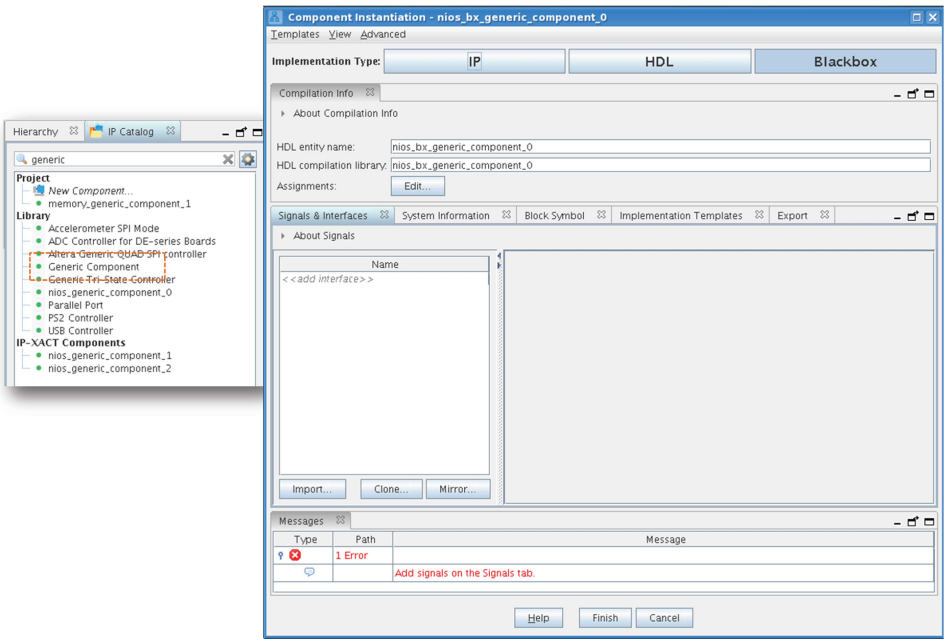
The generic component is a new type of Qsys component that enables hierarchical isolation of IP components. This component is available in the IP Catalog. Use this component as a mechanism to quickly define a custom component or import your RTL into a Qsys Pro system.

By default, the generic component's **Implementation Type** is set to **Blackbox**. This mode specifies that the RTL implementation is not provided in the generated RTL output of the Qsys Pro system. When you generate a system containing a generic component, the system's RTL instantiates the component, but does not provide an implementation for the component. You must provide an implementation for the component in a downstream compiler such as Quartus Prime software or RTL simulators.

To add a generic component to your system:

1. Type `generic component` in the IP Catalog.
2. To launch the **Component Instantiation** editor, double-click **Generic Component**.
3. In the **Compilation Info** tab, specify the **HDL entity name** and **HDL compilation library** name.
4. To add interfaces and signals, click <<*add interface*>> or <<*add signal*>> in the **Signals & Interfaces** tab.
5. To import signals and interfaces from an RTL file, set the **Implementation Type** to **HDL**.
6. To specify the HDL file, click the **Add File** button in the **Files** tab.
7. To import the interfaces from an RTL to the generic component, click **Analyze Synthesis Files**.
8. Click **Finish**. The generic component appears in the **System Contents** tab.

Figure 166. Adding a Generic Component to the Qsys Pro System



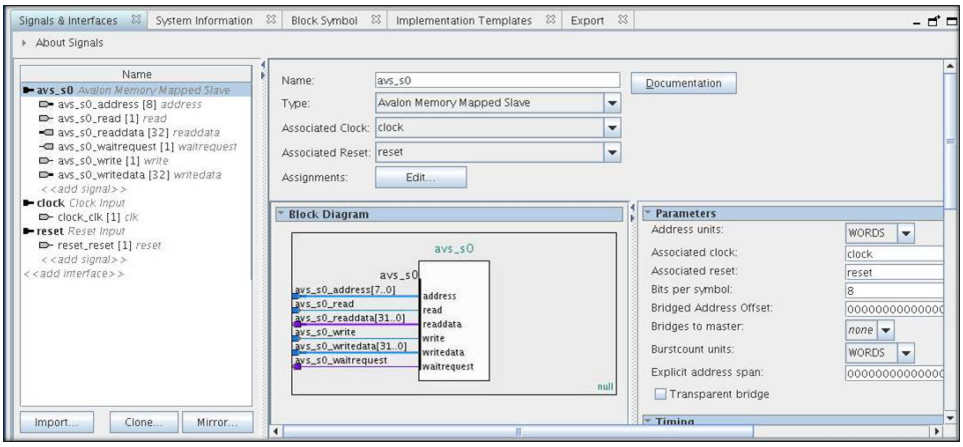
10.14.1 Creating Custom Interfaces in a Generic Component

The **Signals & Interfaces** tab of the **Component Instantiation** editor allows you to customize signals and interfaces for your generic component:

1. Double-click **Generic Component** in the IP Catalog.
2. In the **Component Instantiation** editor, click the **Signals & Interfaces** tab.
3. To add an interface, click <<*add interface*>> in the left pane and select the interface. The selected interface appears in the parameter editor to the right, where you specify its parameters.
4. To add signals to the selected interface, click <<*add signal*>> below the selected interface.
5. To move signals between interfaces, select the signal and drag it to another interface.
6. To rename a signal or interface, select the element, and then press F2.
7. To remove a signal or interface, right-click the element, and then click **Remove**.

   Note: Alternatively, to remove a signal or interface, select the element and press **Delete**. When you remove an interface, Qsys Pro also removes all of its associated signals.

Figure 167. Creating Custom Interfaces



Note: To add existing template interfaces to your generic component, select the interface from **Templates** menu in the **Component Instantiation** editor.

10.14.2 Mirroring Interfaces in a Generic Component

To mirror existing signals and interfaces from an IP component to your generic component:

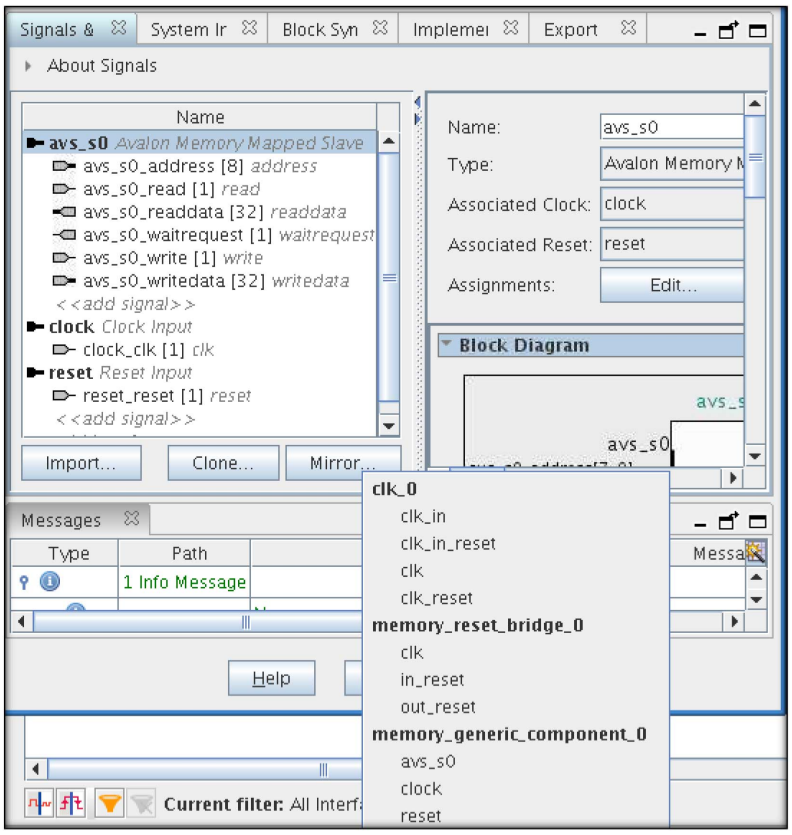
1. Double-click **Generic Component** in the IP Catalog.
2. In the **Component Instantiation** editor, click the **Signals & Interfaces** tab.
3. Click the **Mirror** button. A list appears which lists all the available components in the system and their associated interfaces.
4. Select the desired interface. Qsys Pro mirrors the interface and its associated signals and adds the mirrored interfaces and signals to the **Signals & Interfaces** tab of the generic component.

Example 99. Mirroring Interfaces in a Generic Component Example

Selected Interface	Mirrored Interface
Avalon Memory-Mapped Master (`avalon_master`)	Avalon Memory-Mapped Slave (`avalon_slave`)

Signals of the Selected Interface	Signals of the Mirrored Interface
`waitrequest`(Input 1)	`waitrequest`(Output 1)
`readdata`(Input 32)	`readdata`(Output 32)
`readdatavalid`(Input 1)	`readdatavalid`(Output 1)
`burstcount`(Output 32)	`burstcount`(Input 32)

Figure 168. Mirroring Interfaces

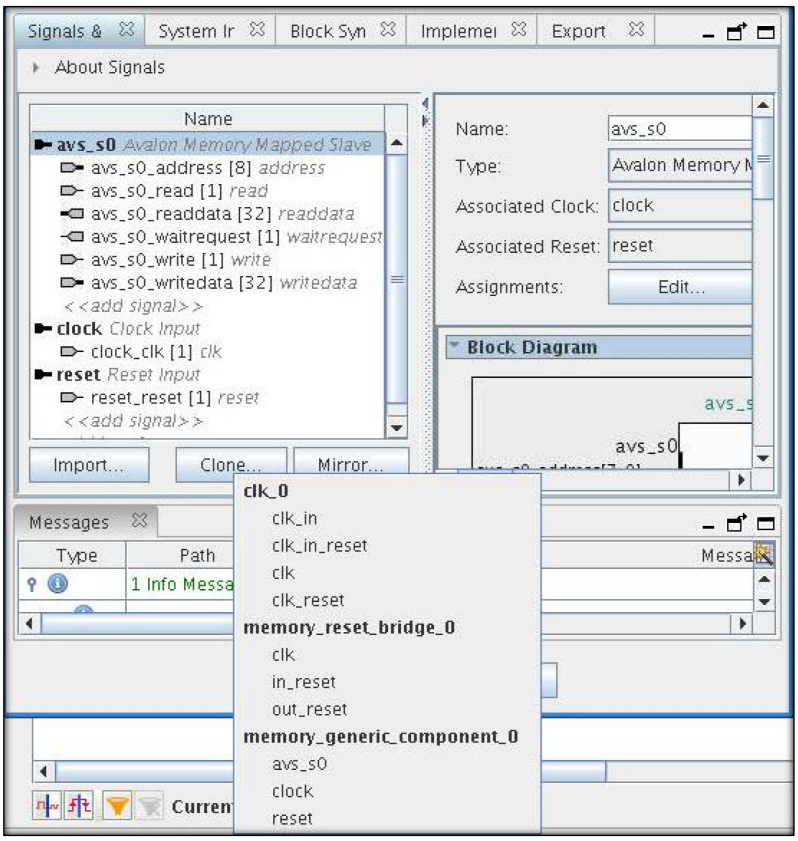


10.14.3 Cloning Interfaces in a Generic Component

To clone existing signals and interfaces from an IP component to your generic component:

1. Double-click **Generic Component** in the IP Catalog.
2. In the **Component Instantiation** editor, click the **Signals & Interfaces** tab.
3. Click the **Clone** button. A list appears which lists all the available components in the system and their associated interfaces.
4. Select the desired interface. Qsys Pro clones the interface and adds an exact replica of the interface and its associated signals to the **Signals & Interfaces** tab of the generic component.

Figure 169. Cloning Interfaces

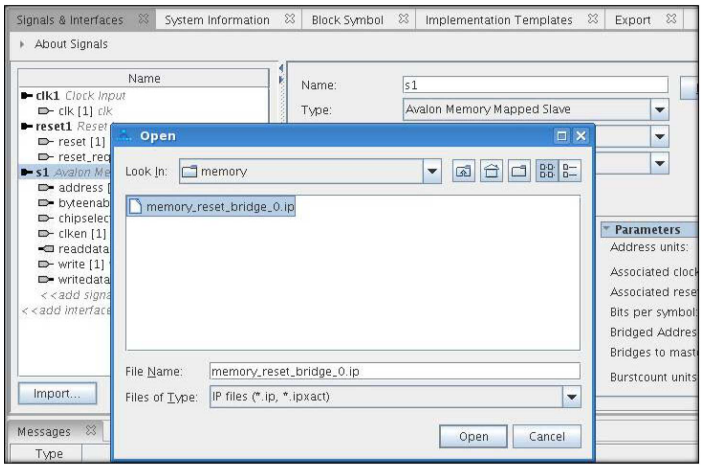


10.14.4 Importing Interfaces to a Generic Component

To import interfaces from an existing IP or IP-XACT file to your generic component:

1. Double-click **Generic Component** in the IP Catalog.
2. In the **Component Instantiation** editor, click the **Signals & Interfaces** tab.
3. Click the **Import** button. A dialog box appears from where you choose the IP/IP-XACT file to import to your generic component.
4. Select the desired interface. Qsys Pro populates the **Signals & Interfaces** tab with the signals and interfaces defined in the selected file.

Figure 170. Importing Interfaces



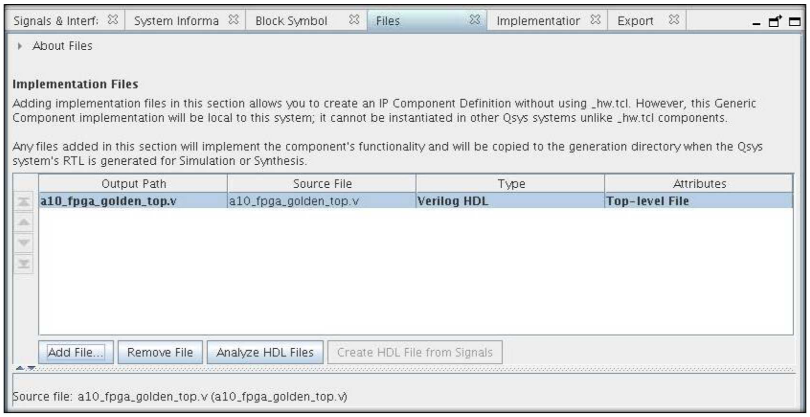
10.14.5 Instantiating RTL in a System as a Generic Component

To add an RTL file as a generic component:

1. Double-click **Generic Component** in the IP Catalog.
2. In the **Component Instantiation** editor, set the **Implementation Type** as **HDL**.
3. Select the **Files** tab.
4. Click **Add File** and select the RTL file to load to the generic component.
5. Click **Analyze HDL files**. This option analyzes and populates the **Signals & Interfaces** tab of the generic component from the RTL file.
6. Verify, and modify the signals and interfaces if needed, in the **Signals & Interfaces** tab.

Note: You must treat a generic component with an **HDL Implementation Type** as a customized and centralized RTL, specific to your current system. When you set a generic component's **Implementation Type** to **HDL**, the output of any RTL that you add to the component is within the system's output directory.

Figure 171. Instantiating an RTL as a Generic Component

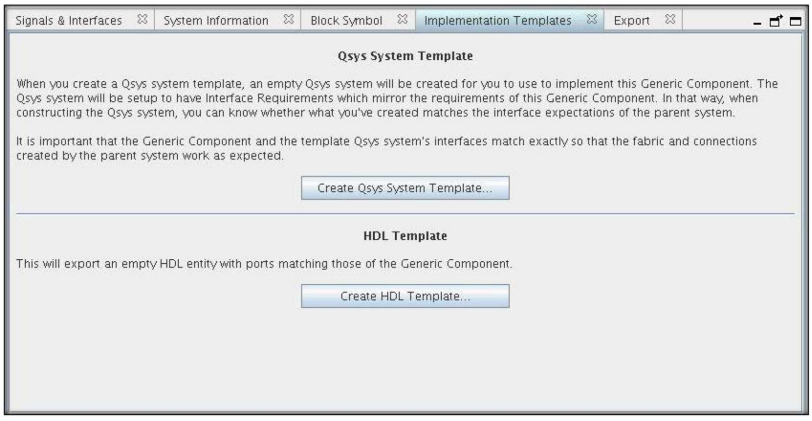


10.14.6 Creating System Template for a Generic Component

To create a Qsys Pro system template:

1. Double-click **Generic Component** in the IP Catalog.
2. In the **Component Instantiation** editor, add the interfaces and signals for the new component in the **Signals & Interfaces** tab.
3. Select the **Implementation Templates** tab.
4. Click **Create Qsys System Template** button. This option creates an empty Qsys Pro system and saves the template as a `.qsys` file to implement this generic component.

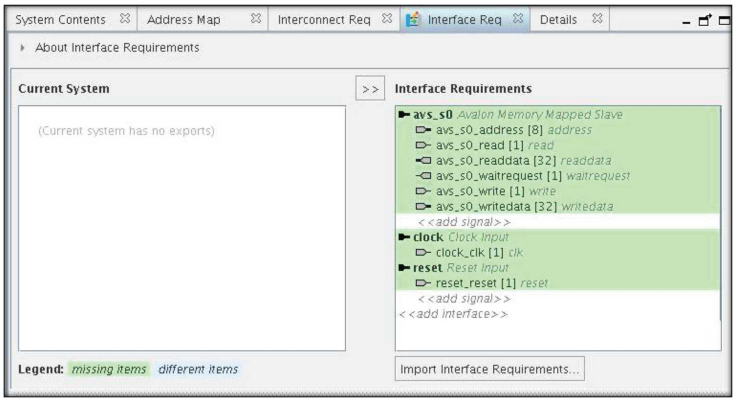
Figure 172. Creating System templates



To implement this component:

1. To open the template Qsys Pro system, click **File ➤ Open** and choose the specific `.qsys` file.
2. Add IP components and/or generic components and export their interfaces to satisfy the specified interface requirements.
3. To view the exported interfaces in the **Interface Requirements** tab, select **View ➤ Interface Requirements**.

Figure 173. Viewing the Interface Requirements from the System Template



10.14.7 Exporting a Generic Component

You can export a generic component as a `.ipxact` file as well as `_hw.tcl` file:

1. Double-click **Generic Component** in the IP Catalog.
2. Select the **Export** tab.
3. To export generic component as an IP-XACT file, click **Export IP-XACT File** and select the location to save your IP-XACT file.
4. To export generic component as a `_hw.tcl` file, click **Export _hw.tcl File** and select the location to save your `_hw.tcl` file.

10.15 Document Revision History

The table below indicates edits made to the *Creating Qsys Pro Components* content since its creation.

Table 117. Document Revision History

Date	Version	Changes
2016.10.31	16.1.0	• Implemented Intel rebranding. • Implemented Qsys Pro rebranding. • Added topics for Generic Component.
2015.11.02	15.1.0	Changed instances of *Quartus II* to *Quartus Prime*.
2015.05.04	15.0.0	• Updated screen shots **Files** tab, Qsys Pro Component Editor. • Added topic: *Specify Interfaces and Signals in the Qsys Pro Component Editor*. • Added topic: *Create an HDL File in the Qsys Pro Component Editor*. • Added topic: *Create an HDL File Using a Template in the Qsys Pro Component Editor*.
November 2013	13.1.0	• `add_hdl_instance` • Added *Creating a Component With Differing Structural Qsys Pro View and Generated Output Files*.
May 2013	13.0.0	• Consolidated content from other Qsys Pro chapters. • Added *Upgrading IP Components to the Latest Version*. • Updated for AMBA APB support.
November 2012	12.1.0	• Added AMBA AXI4 support. • Added the **demo_axi_memory** example with screen shots and example `_hw.tcl` code.
June 2012	12.0.0	• Added new tab structure for the Component Editor. • Added AMBA AXI3 support.
November 2011	11.1.0	Template update.
May 2011	11.0.0	• Removed beta status. • Added Avalon Tri-state Conduit (Avalon-TC) interface type. • Added many interface templates for Nios custom instructions and Avalon-TC interfaces.
December 2010	10.1.0	Initial release.

Related Links

Altera Documentation Archive

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

11 Qsys Pro Interconnect

Qsys Pro interconnect is a high-bandwidth structure that allows you to connect IP components to other IP components with various interfaces.

Qsys Pro supports Avalon, AMBA AXI3 (version 1.0), AMBA AXI4 (version 2.0), AMBA AXI4-Lite (version 2.0), AMBA AXI4-Stream (version 1.0), and AMBA APB3 (version 1.0) interface specifications.

Note: The video, *AMBA AXI and Altera Avalon Interoperation Using Qsys Pro*, describes seamless integration of IP components using the AMBA AXI interface, and the Altera Avalon interface.

Related Links

- Avalon Interface Specifications
- AMBA Specifications
- Creating a System with Qsys Pro on page 299
  Qsys Pro is a system integration tool included as part of the Quartus Prime software.
- Creating Qsys Pro Components on page 585
  You can create a Hardware Component Definition File (`_hw.tcl`) to describe and package IP components for use in a Qsys Pro system.
- Qsys Pro System Design Components on page 879
  You can use Qsys Pro IP components to create Qsys Pro systems.
- AMBA AXI and Altera Avalon Interoperation Using Qsys Pro

11.1 Memory-Mapped Interfaces

Qsys Pro supports the implementation of memory-mapped interfaces for Avalon, AXI, and APB protocols.

Qsys Pro interconnect transmits memory-mapped transactions between masters and slaves in packets. The command network transports read and write packets from master interfaces to slave interfaces. The response network transports response packets from slave interfaces to master interfaces.

For each component interface, Qsys Pro interconnect manages memory-mapped transfers and interacts with signals on the connected interface. Master and slave interfaces can implement different signals based on interface parameterizations, and Qsys Pro interconnect provides any necessary adaptation between them. In the path between master and slaves, Qsys Pro interconnect may introduce registers for timing synchronization, finite state machines for event sequencing, or nothing at all, depending on the services required by the interfaces.

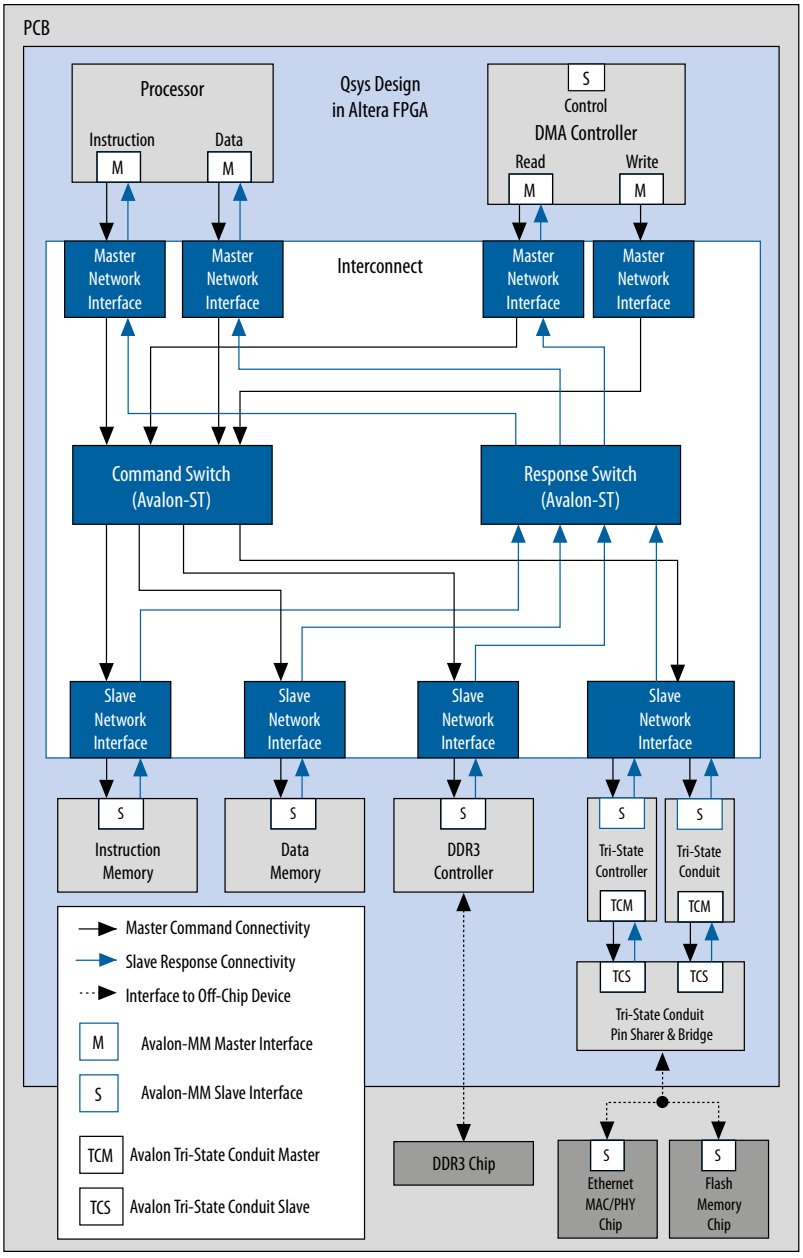
Qsys Pro interconnect supports the following implementation scenarios:

- Any number of components with master and slave interfaces. The master-to-slave relationship can be one-to-one, one-to-many, many-to-one, or many-to-many.
- Masters and slaves of different data widths.
- Masters and slaves operating in different clock domains.
- IP Components with different interface properties and signals. Qsys Pro adapts the component interfaces so that interfaces with the following differences can be connected:
  - Avalon and AXI interfaces that use active-high and active-low signaling. AXI signals are active high, except for the reset signal.
  - Interfaces with different burst characteristics.
  - Interfaces with different latencies.
  - Interfaces with different data widths.
  - Interfaces with different optional interface signals.

    Note: AXI3/4 to AXI3/4 interface connections declare a fixed set of signals with variable latency. As a result, there is no need for adapting between active-low and active-high signaling, burst characteristics, different latencies, or port signatures. Some adaptation may be necessary between Avalon interfaces.

Figure 174. Qsys Pro interconnect for an Avalon-MM System with Multiple Masters

In this example, there are two components mastering the system, a processor and a DMA controller, each with two master interfaces. The masters connect through the Qsys Pro interconnect to several slaves in the Qsys Pro system. The dark blue blocks represent interconnect components. The dark grey boxes indicate items outside of the Qsys Pro system and the Quartus Prime software design, and show how component interfaces can be exported and connected to external devices.



11.1.1 Qsys Pro Packet Format

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

11.1.1.1 Qsys Pro Packet Format

Table 118. Qsys Pro Packet Format for Memory-Mapped Master and Slave Interfaces

The fields of the Qsys Pro packet format are of variable length to minimize resource usage. However, if the majority of components in a design have a single data width, for example 32-bits, and a single component has a data width of 64-bits, Qsys Pro inserts a width adapter to accommodate 64-bit transfers.

Command	Description
`Address`	Specifies the byte address for the lowest byte in the current cycle. There are no restrictions on address alignment.
`Size`	Encodes the run-time size of the transaction. In conjunction with address, this field describes the segment of the payload that contains valid data for a beat within the packet.
`Address Sideband`	Carries "address" sideband signals. The interconnect passes this field from master to slave. This field is valid for each beat in a packet, even though it is only produced and consumed by an address cycle. Up to 8-bit sideband signals are supported for both read and write address channels.
`Cache`	Carries the AXI cache signals.
`Transaction (Exclusive)`	Indicates whether the transaction has exclusive access.
`Transaction (Posted)`	Used to indicate non-posted writes (writes that require responses).
`Data`	For command packets, carries the data to be written. For read response packets, carries the data that has been read.
`Byteenable`	Specifies which symbols are valid. AXI can issue or accept any byteenable pattern. For compatibility with Avalon, Intel recommends that you use the following legal values for 32-bit data transactions between Avalon masters and slaves: • **1111**—Writes full 32 bits • **0011**—Writes lower 2 bytes • **1100**—Writes upper 2 bytes • **0001**—Writes byte 0 only • **0010**—Writes byte 1 only • **0100**—Writes byte 2 only • **1000**—Writes byte 3 only
`Source_ID`	The ID of the master or slave that initiated the command or response.
`Destination_ID`	The ID of the master or slave to which the command or response is directed.
`Response`	Carries the AXI response signals.
`Thread ID`	Carries the AXI transaction ID values.
`Byte count`	The number of bytes remaining in the transaction, including this beat. Number of bytes requested by the packet.

continued...

Command	Description
Burstwrap	The burstwrap value specifies the wrapping behavior of the current burst. The burstwrap value is of the form $2^{<n>}$ -1. The following types are defined: • Variable wrap–Variable wrap bursts can wrap at any integer power of 2 value. When the burst reaches the wrap boundary, it wraps back to the previous burst boundary so that only the low order bits are used for addressing. For example, a burst starting at address 0x1C, with a burst wrap boundary of 32 bytes and a burst size of 20 bytes, would write to addresses 0x1C, 0x0, 0x4, 0x8, and 0xC. • For a burst wrap boundary of size <*m*>, Burstwrap = <*m*> - 1, or for this case Burstwrap = (32 - 1) = 31 which is 2^5 -1. • For AXI masters, the burstwrap boundary value (m) is based on the different AXBURST: — Burstwrap set to all 1's. For example, for a 6-bit burstwrap, burstwrap is 6'b111111. — For WRAP bursts, burstwrap = AXLEN * size – 1. — For FIXED bursts, burstwrap = size – 1. — Sequential bursts increment the address for each transfer in the burst. For sequential bursts, the Burstwrap field is set to all 1s. For example, with a 6-bit Burstwrap field, the value for a sequential burst is 6'b111111 or 63, which is 2^6 - 1. For Avalon masters, Qsys Pro adaptation logic sets a hardwired value for the burstwrap field, according the declared master burst properties. For example, for a master that declares sequential bursting, the burstwrap field is set to ones. Similarly, masters that declare burst have their burstwrap field set to the appropriate constant value. AXI masters choose their burst type at run-time, depending on the value of the AW or ARBURST signal. The interconnect calculates the burstwrap value at run-time for AXI masters.
Protection	Access level protection. When the lowest bit is 0, the packet has normal access. When the lowest bit is 1, the packet has privileged access. For Avalon-MM interfaces, this field maps directly to the privileged access signal, which allows a memory-mapped master to write to an on-chip memory ROM instance. The other bits in this field support AXI secure accesses and uses the same encoding, as described in the AXI specification.
QoS	QoS (Quality of Service Signaling) is a 4-bit field that is part of the AXI4 interface that carries QoS information for the packet from the AXI master to the AXI slave. Transactions from AXI3 and Avalon masters have the default value 4'b0000, that indicates that they are not participating in the QoS scheme. QoS values are dropped for slaves that do not support QoS.
Data sideband	Carries data sideband signals for the packet. On a write command, the data sideband directly maps to WUSER. On a read response, the data sideband directly maps to RUSER. On a write response, the data sideband directly maps to BUSER.

11.1.1.2 Transaction Types for Memory-Mapped Interfaces

Table 119. Transaction Types for Memory-Mapped Interfaces

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

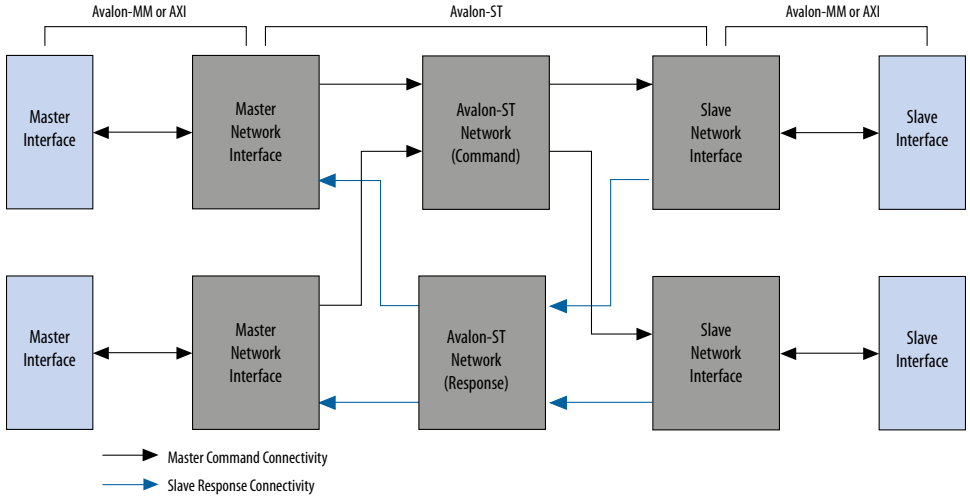
Bit	Name	Definition
0	PKT_TRANS_READ	When asserted, indicates a read transaction.
1	PKT_TRANS_COMPRESSED_READ	For read transactions, specifies whether or not the read command can be expressed in a single cycle, that is whether or not it has all byteenables asserted on every cycle.
2	PKT_TRANS_WRITE	When asserted, indicates a write transaction.
3	PKT_TRANS_POSTED	When asserted, no response is required.
4	PKT_TRANS_LOCK	When asserted, indicates arbitration is locked. Applies to write packets.

11.1.1.3 Qsys Pro Transformations

The memory-mapped master and slave components connect to network interface modules that encapsulate the transaction in Avalon-ST packets. The memory-mapped interfaces have no information about the encapsulation or the function of the layer transporting the packets. The interfaces operate in accordance with memory-mapped protocol and use the read and write signals and transfers.

Figure 175. Transformation when Generating a System with Memory-Mapped and Slave Components

Qsys Pro components that implement the blocks appear shaded.



Related Links

- Master Network Interfaces on page 634
- Slave Network Interfaces on page 637

11.1.2 Interconnect Domains

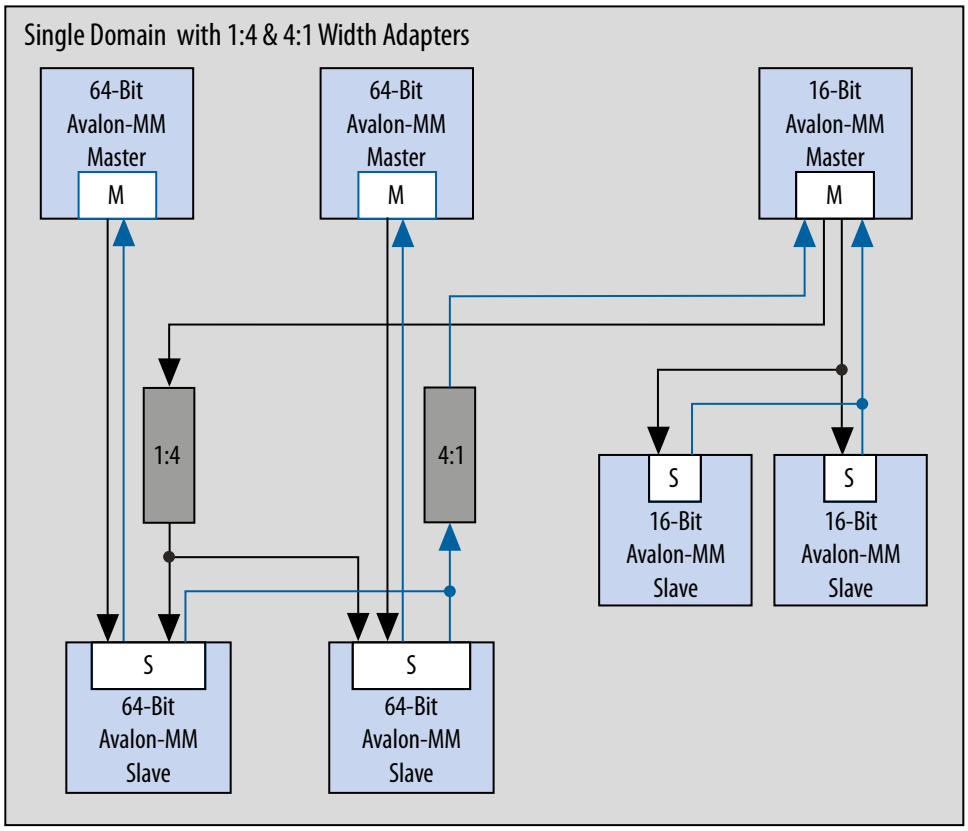
An interconnect domain is a group of connected memory-mapped masters and slaves that share the same interconnect.The components in a single interconnect domain share the same packet format.

11.1.2.1 Using One Domain with Width Adaptation

When one of the masters in a system connects to all of the slaves, Qsys Pro creates a single domain with two packet formats: one with 64-bit data, and one with 16-bit data. A width adapter manages accesses between the 16-bit master and 64-bit slaves.

Figure 176. One Domain with 1:4 and 4:1 Width Adapters

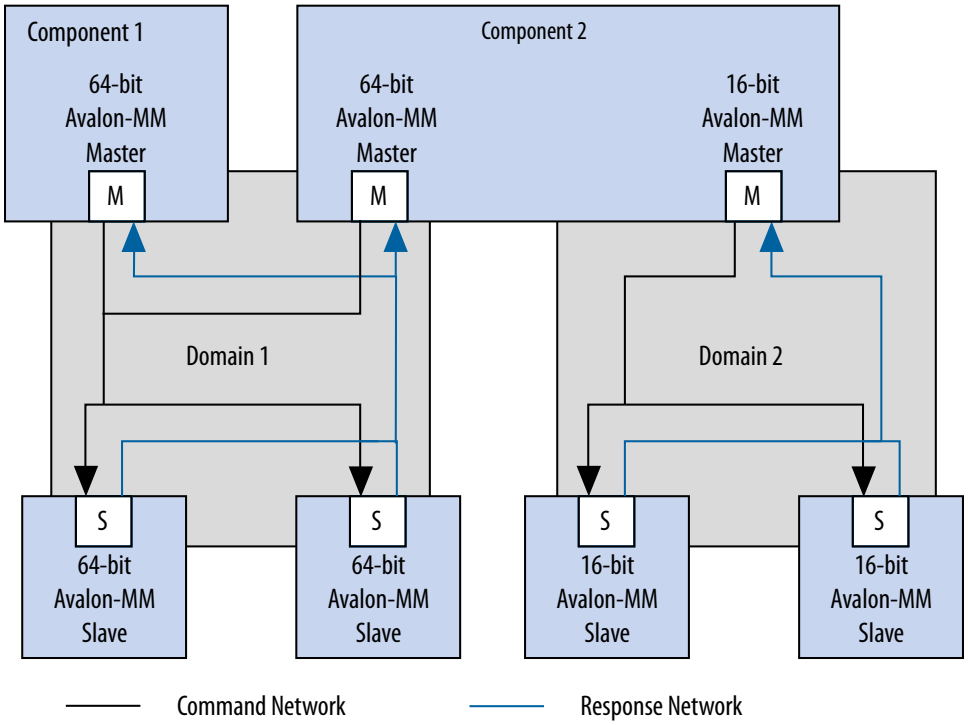
In this system example, there are two 64-bit masters that access two 64-bit slaves. It also includes one 16-bit master, that accesses two 16-bit slaves and two 64-bit slaves. The 16-bit Avalon master connects through a 1:4 adapter, then a 4:1 adapter to reach its 16-bit slaves.



11.1.2.2 Using Two Separate Domains

Figure 177. Two Separate Domains

In this system example, Qsys Pro uses two separate domains. The first domain includes two 64-bit masters connected to two 64-bit slaves. A second domain includes one 16-bit master connected to two 16-bit slaves. Because the interfaces in Domain 1 and Domain 2 do not share any connections, Qsys Pro can optimize the packet format for the two separate domains. In this example, the first domain uses a 64-bit data width and the second domain uses 16-bit data.



11.1.3 Master Network Interfaces

Figure 178. Avalon-MM Master Network Interface

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

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

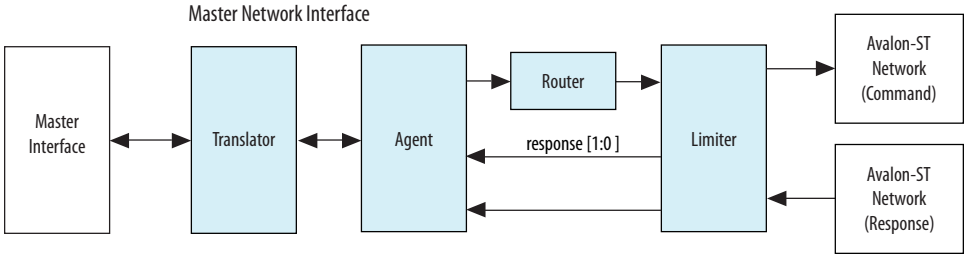
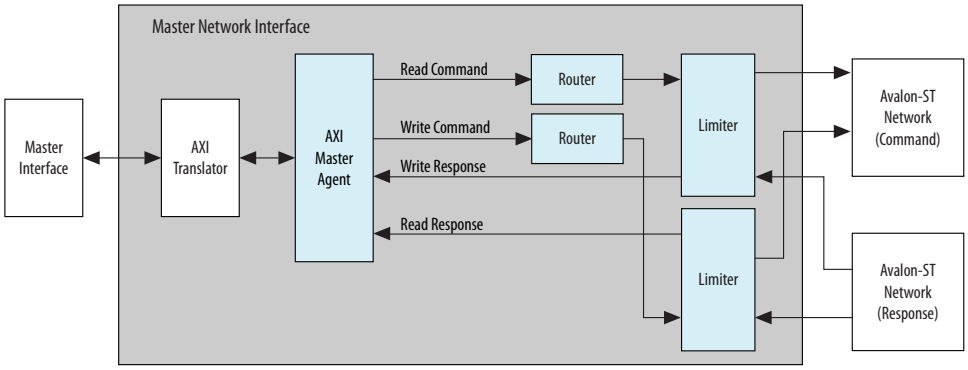


Figure 179. AXI Master Network Interface

An AXI4 master supports INCR bursts up to 256 beats, QoS signals, and data sideband signals.



Note: For a complete definition of the optional read response signal, refer to *Avalon Memory-Mapped Interface Signal Types* in the *Avalon Interface Specifications*.

Related Links

- Read and Write Responses on page 650
- Avalon Interface Specifications
- Creating a System with Qsys Pro on page 299
  Qsys Pro is a system integration tool included as part of the Quartus Prime software.

11.1.3.1 Avalon-MM Master Agent

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

11.1.3.2 Avalon-MM Master Translator

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

The Avalon-MM Master translator performs the following functions:

- Translates active-low signaling to active-high signaling
- Inserts wait states to prevent an Avalon-MM master from reading invalid data
- Translates word and symbol addresses
- Translates word and symbol burst counts
- Manages re-timing and re-sequencing bursts
- Removes unnecessary address bits

11.1.3.3 AXI Master Agent

An AXI Master Agent accepts AXI commands and produces Qsys Pro command packets. It also accepts Qsys Pro response packets and converts those into AXI responses. This component has separate packet channels for read commands, write commands, read responses, and write responses. Avalon master agent drives the QoS and BUSER, WUSER, and RUSER packet fields with default values AXQO and b0000, respectively.

Note: For signal descriptions, refer to *Qsys Pro Packet Format*.

Related Links

Qsys Pro Packet Format on page 630

11.1.3.4 AXI Translator

AXI4 allows some signals to be omitted from interfaces. The translator bridges between these "incomplete" AXI4 interfaces and the "complete" AXI4 interface on the network interfaces.

The AXI translator is inserted for both AXI4 masters and slaves and performs the following functions:

- Matches ID widths between the master and slave in 1x1 systems.
- Drives default values as defined in the *AMBA Protocol Specifications* for missing signals.
- Performs lock transaction bit conversion when an AXI3 master connects to an AXI4 slave in 1x1 systems.

Related Links

AMBA Protocol Specifications

11.1.3.5 APB Master Agent

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

11.1.3.6 APB Slave Agent

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

11.1.3.7 APB Translator

An APB peripheral does not require pslverr signals to support additional signals for the APB debug interface.

The APB translator is inserted for both the master and slave and performs the following functions:

- Sets the response value default to OKAY if the APB slave does not have a pslverr signal.
- Turns on or off additional signals between the APB debug interface, which is used with HPS (Intel SoC's Hard Processor System).

11.1.3.8 AHB Slave Agent

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

11.1.3.9 Memory-Mapped Router

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

11.1.3.10 Memory-Mapped Traffic Limiter

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

11.1.4 Slave Network Interfaces

Figure 180. Avalon-MM Slave Network Interface

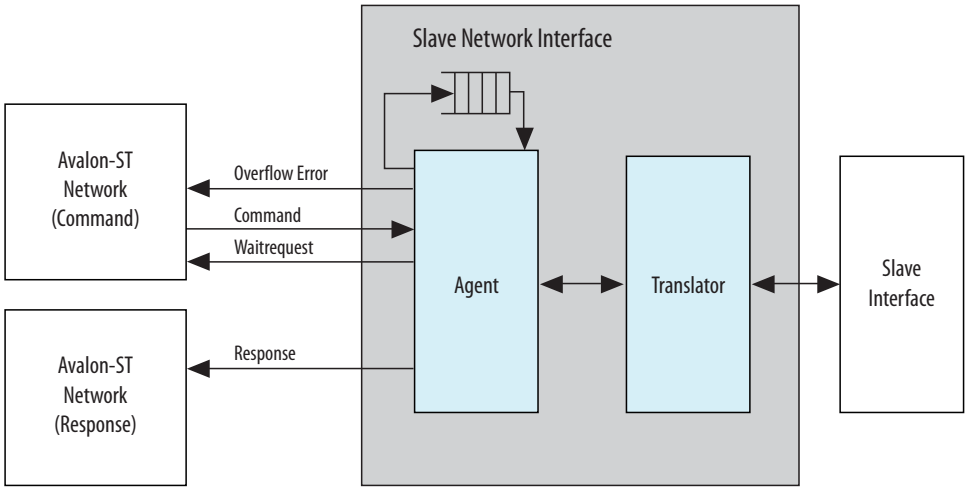
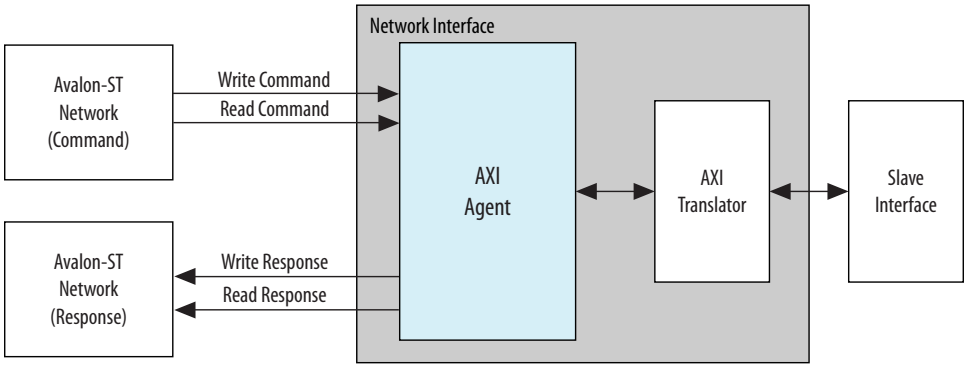


Figure 181. AXI Slave Network Interface

An AXI4 slave supports up to 256 beat INCR bursts, QoS signals, and data sideband signals.



11.1.4.1 Avalon-MM Slave Translator

The Avalon-MM Slave Translator interfaces to an Avalon-MM slave component as the *Avalon-MM Slave Network Interface* figure illustrates. It converts the Avalon-MM slave interface to a simplified representation that the Qsys Pro network can use.

An Avalon-MM Merlin Slave Translator performs the following functions:

- Drives the `beginbursttransfer` and `byteenable` signals.
- Supports Avalon-MM slaves that operate using fixed timing and or slaves that use the `readdatavalid` signal to identify valid data.
- Translates the `read`, `write`, and `chipselect` signals into the representation that the Avalon-ST slave response network uses.
- Converts active low signals to active high signals.
- Translates word and symbol addresses and burstcounts.
- Handles burstcount timing and sequencing.
- Removes unnecessary address bits.

Related Links

Slave Network Interfaces on page 637

11.1.4.2 AXI Translator

AXI4 allows some signals to be omitted from interfaces. The translator bridges between these "incomplete" AXI4 interfaces and the "complete" AXI4 interface on the network interfaces.

The AXI translator is inserted for both AXI4 master and slave, and performs the following functions:

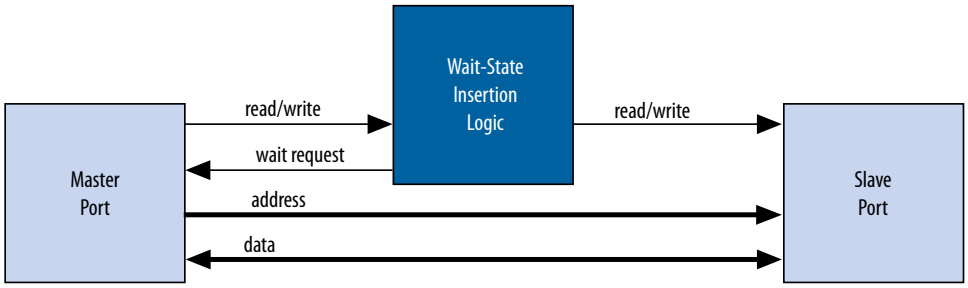
- Matches ID widths between master and slave in 1x1 systems.
- Drives default values as defined in the *AMBA Protocol Specifications* for missing signals.
- Performs lock transaction bit conversion when an AXI3 master connects to an AXI4 slave in 1x1 systems.

11.1.4.3 Wait State Insertion

Wait states extend the duration of a transfer by one or more cycles. Wait state insertion logic accommodates the timing needs of each slave, and causes the master to wait until the slave can proceed. Qsys Pro interconnect inserts wait states into a transfer when the target slave cannot respond in a single clock cycle, as well as in cases when slave `read` and `write` signals have setup or hold time requirements.

Figure 182. Wait State Insertion Logic for One Master and One Slave

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



11.1.4.4 Avalon-MM Slave Agent

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

11.1.4.5 AXI Slave Agent

An AXI Slave Agent works similar to a master agent in reverse. The AXI slave Agent accepts Qsys Pro command packets to create AXI commands, and accepts AXI responses to create Qsys Pro response packets. This component has separate packet channels for read commands, write commands, read responses, and write responses.

11.1.5 Arbitration

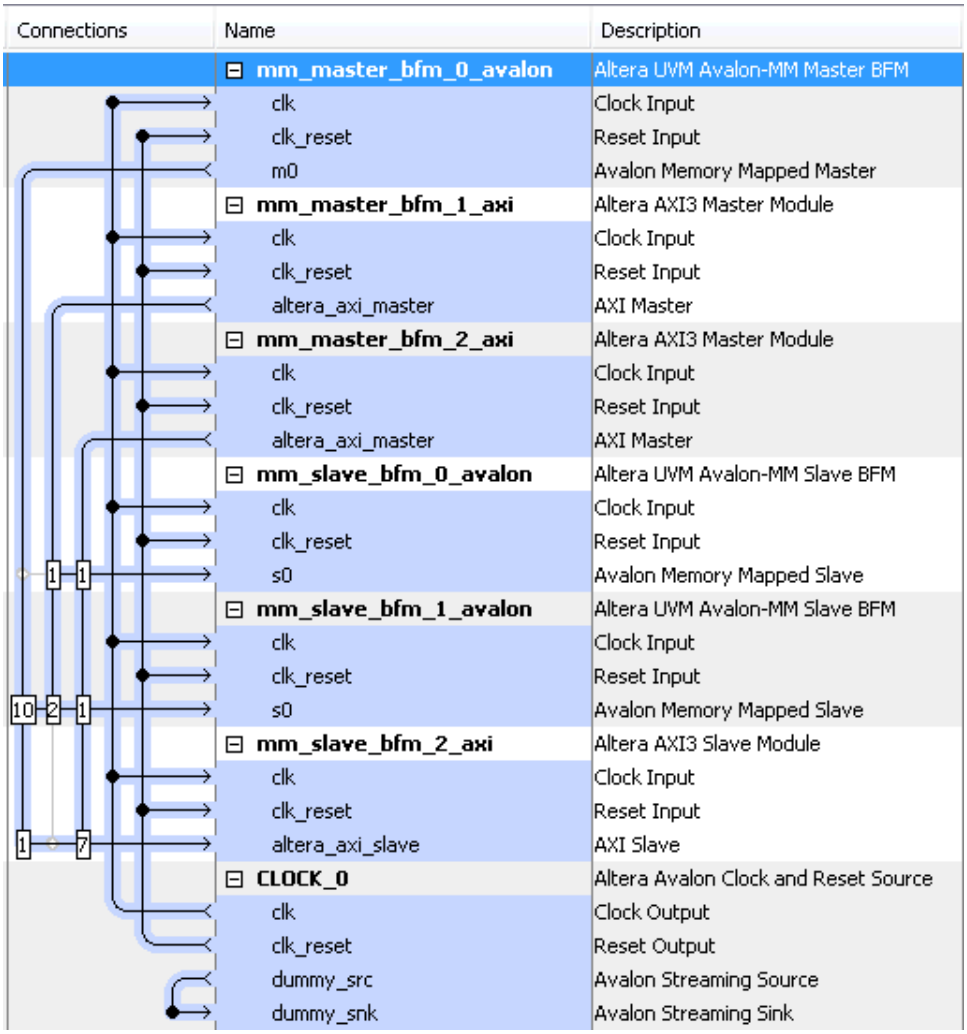
When multiple masters contend for access to a slave, Qsys Pro automatically inserts arbitration logic, which grants access in fairness-based, round-robin order. You can alternatively choose to designate a slave as a fixed priority arbitration slave, and then manually assign priorities in the Qsys Pro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

11.1.5.1 Round-Robin Arbitration

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

In a fairness-based arbitration protocol, each master has an integer value of transfer *shares* with respect to a slave. One share represents permission to perform one transfer. The default arbitration scheme is equal share round-robin that grants equal, sequential access to all requesting masters. You can change the arbitration scheme to weighted round-robin by specifying a relative number of arbitration shares to the masters that access a particular slave. AXI slaves have separate arbitration for their independent read and write channels, and the **Arbitration Shares** setting affects both the read and write arbitration. To display arbitration settings, right-click an instance on the **System Contents** tab, and then click **Show Arbitration Shares**.

Figure 183. Arbitration Shares in the Connections Column



11.1.5.1.1 Fairness-Based Shares

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

Figure 184. Arbitration of Continuous Transfer Requests from Two Masters

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

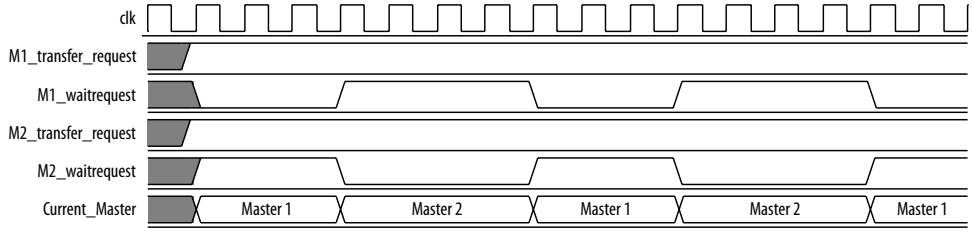
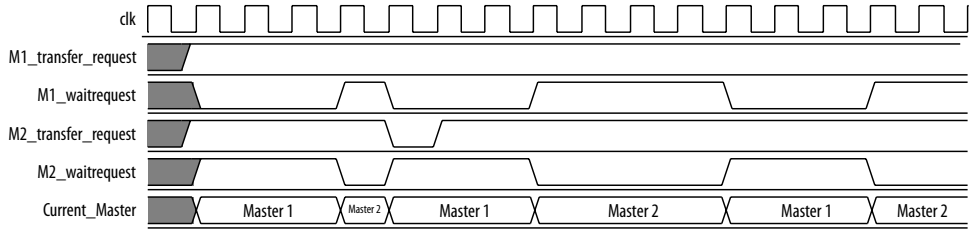


Figure 185. Arbitration of Two Masters with a Gap in Transfer Requests

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



11.1.5.1.2 Round-Robin Scheduling

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

11.1.5.2 Fixed Priority Arbitration

Fixed priority arbitration is an alternative arbitration scheme to the default round-robin arbitration scheme.

You can selectively apply fixed priority arbitration to any slave in a Qsys Pro system. You can design Qsys Pro systems where some slaves use the default round-robin arbitration, and other slaves use fixed priority arbitration. Fixed priority arbitration uses a fixed priority algorithm to grant access to a slave amongst its connected masters.

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

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

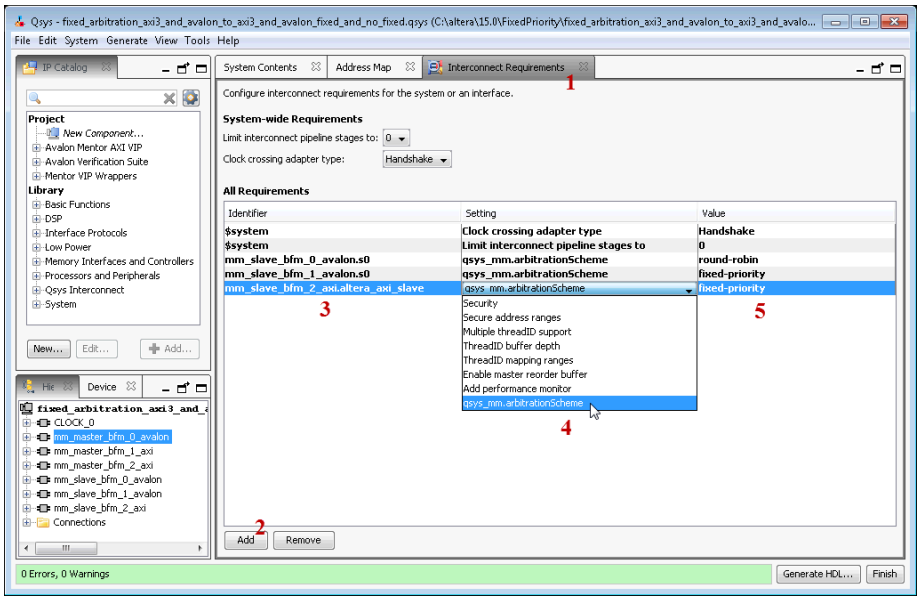
Note: When you connect an AXI master to an Avalon-MM slave designated to use a fixed priority arbitrator, the interconnect instantiates a command-path intermediary round-robin multiplexer in front of the designated slave.

11.1.5.2.1 Designate a Qsys Pro Slave to Use Fixed Priority Arbitration

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

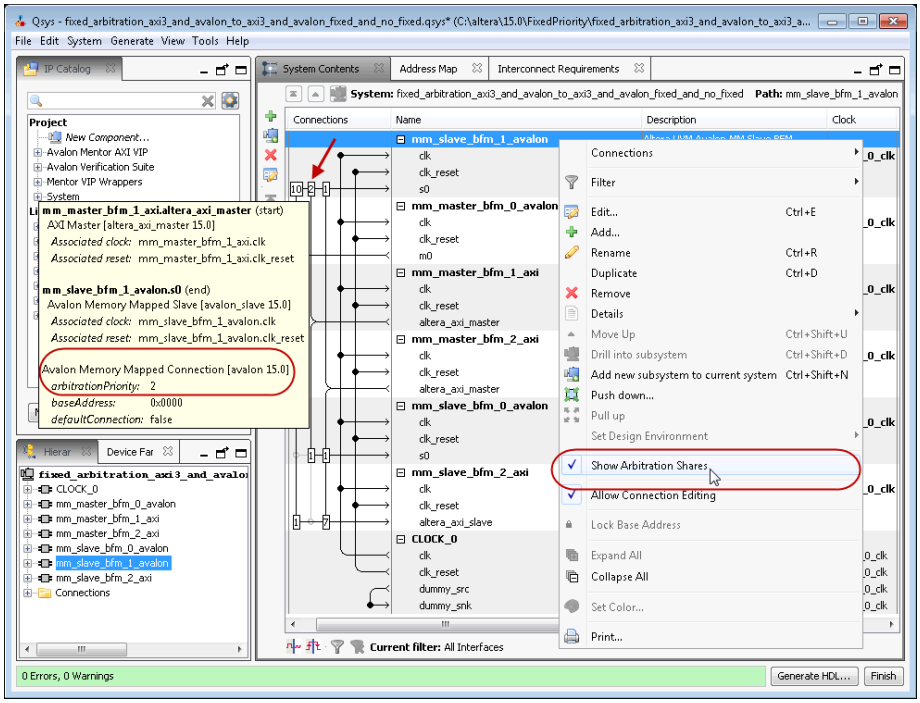
1. In Qsys Pro, navigate to the **Interconnect Requirements** tab.
2. Click **Add** to add a new requirement.
3. In the **Identifier** column, select the slave for fixed priority arbitration.
4. In the **Setting** column, select **qsys mm.arbitrationScheme**.
5. In the **Value** column, select **fixed-priority**.

Figure 186. Designate a Slave to Use Fixed Priority Arbitration



6. Navigate to the **System Contents** tab.
7. In the **System Contents** tab, right-click the designated fixed priority slave, and then select **Show Arbitration Shares**.
8. For each master connected to the fixed priory arbitration slave, type a numerical arbitration priority in the box that appears in place of the connection circle.

Figure 187. Arbitration Priorities in the Qsys Pro System Contents Tab



9. Right click the designated fixed priority slave and uncheck **Show Arbitration Shares** to return to the connection circles.

11.1.5.2.2 Fixed Priority Arbitration with AXI Masters and Avalon-MM Slaves

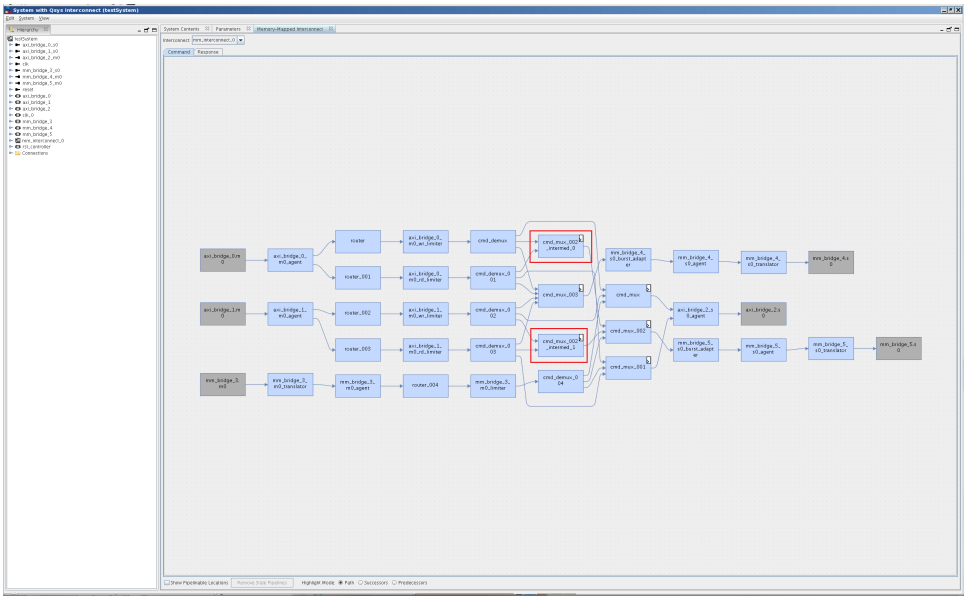
When an AXI master is connected to a designated fixed priority arbitration Avalon-MM slave, Qsys Pro interconnect automatically instantiates an intermediary multiplexer in front of the Avalon-MM slave.

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

When an AXI master is connected to a fixed priority AXI slave, the master's read and write channels are directly connected to the AXI slave's fixed-priority multiplexers. In this case, there is one multiplexer for the read command, and one multiplexer for the write command and therefore an intermediary multiplexer is not required.

The red circles indicate placement of the intermediary multiplexer between the AXI master and Avalon-MM slave due to the separate read and write channels of the AXI master.

Figure 188. Intermediary Multiplexer Between AXI Master and Avalon-MM Slave

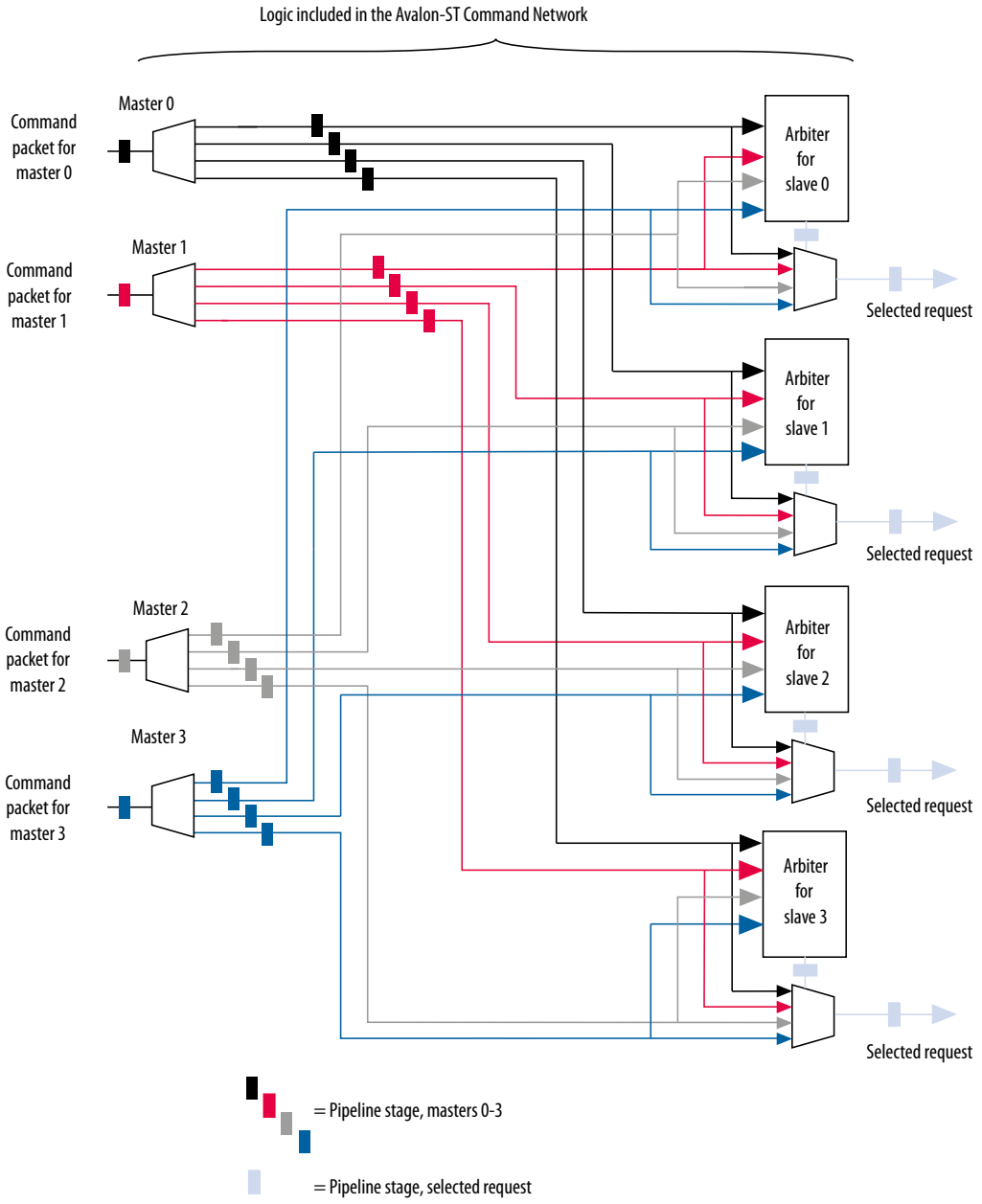


11.1.6 Memory-Mapped Arbiter

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

Figure 189. Arbitration Logic

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



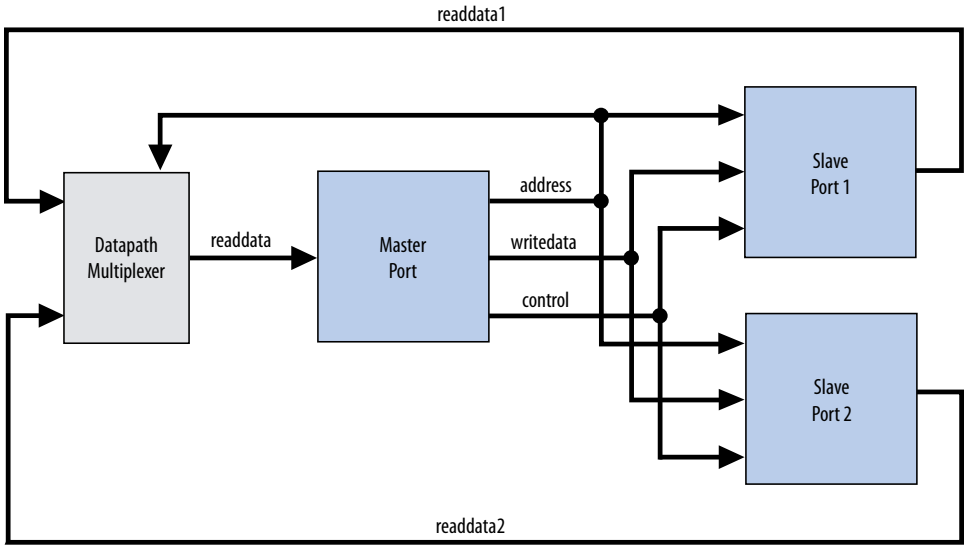
Note: If you specify a **Limit interconnect pipeline stages to** parameter greater than zero, the output of the Arbiter is registered. Registering this output reduces the amount of combinational logic between the master and the interconnect, increasing the f_{MAX} of the system.

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

11.1.7 Datapath Multiplexing Logic

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

Figure 190. Datapath Multiplexing Logic for One Master and Two Slaves



11.1.8 Width Adaptation

Qsys Pro width adaptation converts between Avalon memory-mapped master and slaves with different data and byte enable widths, and manages the run-time size requirements of AXI. Width adaptation for AXI to Avalon interfaces is also supported.

11.1.8.1 Memory-Mapped Width Adapter

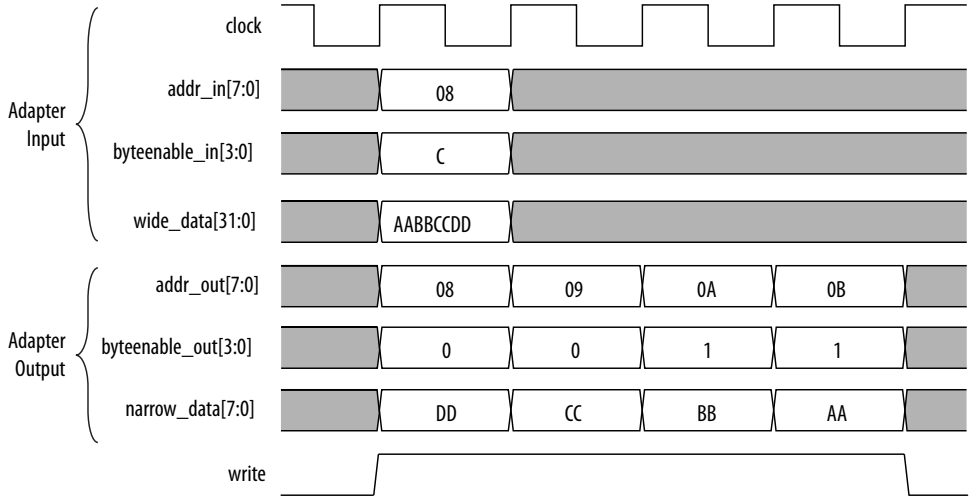
The Memory-Mapped Width Adapter is used in the Avalon-ST domain and operates with information contained in the packet format.

The memory-mapped width adapter accepts packets on its sink interface with one data width and produces output packets on its source interface with a different data width. The ratio of the narrow data width must be a power of two, such as 1:4, 1:8, and 1:16. The ratio of the wider data width to the narrower width must also be a power of two, such as 4:1, 8:1, and 16:1 These output packets may have a different size if the input size exceeds the output data bus width, or if data packing is enabled.

When the width adapter converts from narrow data to wide data, each input beat's data and byte enables are copied to the appropriate segment of the wider output data and byte enables signals.

Figure 191. Width Adapter Timing for a 4:1 Adapter

This adapter assumes that the field ordering of the input and output packets is the same, with the only difference being the width of the data and accompanying byte enable fields. When the width adapter converts from wide data to narrow data, the narrower data is transmitted over several beats. The first output beat contains the lowest addressed segment of the input data and byte enables.



11.1.8.1.1 AXI Wide-to-Narrow Adaptation

For all cases of AXI wide-to-narrow adaptation, read data is re-packed to match the original size. Responses are merged, with the following error precedence: DECERR, SLVERR, 0KAY, and EX0KAY.

Table 120. AXI Wide-to-Narrow Adaptation (Downsizing)

Burst Type	Behavior
Incrementing	If the transaction size is less than or equal to the output width, the burst is unmodified. Otherwise, it is converted to an incrementing burst with a larger length and size equal to the output width. If the resulting burst is unsuitable for the slave, the burst is converted to multiple sequential bursts of the largest allowable lengths. For example, for a 2:1 downsizing ratio, an `INCR9` burst is converted into `INCR16 + INCR2` bursts. This is true if the maximum burstcount a slave can accept is 16, which is the case for AXI3 slaves. Avalon slaves have a maximum burstcount of 64.
Wrapping	If the transaction size is less than or equal to the output width, the burst is unmodified. Otherwise, it is converted to a wrapping burst with a larger length, with a size equal to the output width. If the resulting burst is unsuitable for the slave, the burst is converted to multiple sequential bursts of the largest allowable lengths; respecting wrap boundaries. For example, for a 2:1 downsizing ratio, a `WRAP16` burst is converted into two or three `INCR` bursts, depending on the address.
Fixed	If the transaction size is less than or equal to the output width, the burst is unmodified. Otherwise, it is converted into repeated sequential bursts over the same addresses. For example, for a 2:1 downsizing ratio, a FIXED single burst is converted into an `INCR2` burst.

11.1.8.1.2 AXI Narrow-to-Wide Adaptation

Table 121. AXI Narrow-to-Wide Adaptation (Upsizing)

Burst Type	Behavior
Incrementing	The burst (and its response) passes through unmodified. Data and write strobes are placed in the correct output segment.
Wrapping	The burst (and its response) passes through unmodified.
Fixed	The burst (and its response) passes through unmodified.

11.1.9 Burst Adapter

Qsys Pro interconnect uses the memory-mapped burst adapter to accommodate the burst capabilities of each interface in the system, including interfaces that do not support burst transfers.

The maximum burst length for each interface is a property of the interface and is independent of other interfaces in the system. Therefore, a particular master may be capable of initiating a burst longer than a slave's maximum supported burst length. In this case, the burst adapter translates the large master burst into smaller bursts, or into individual slave transfers if the slave does not support bursting. Until the master completes the burst, arbiter logic prevents other masters from accessing the target slave. For example, if a master initiates a burst of 16 transfers to a slave with maximum burst length of 8, the burst adapter initiates 2 bursts of length 8 to the slave.

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

Note: AXI masters can issue burst types that Avalon cannot accept, for example, fixed bursts. In this case, the burst adapter converts the fixed burst into a sequence of transactions to the same address.

Note: For AXI4 slaves, Qsys Pro allows 256-beat `INCR` bursts. You must ensure that 256-beat narrow-sized `INCR` bursts are shortened to 16-beat narrow-sized `INCR` bursts for AXI3 slaves.

Avalon-MM masters always issue addresses that are aligned to the size of the transfer. However, when Qsys Pro uses a narrow-to-wide width adaptation, the resulting address may be unaligned. For unaligned addresses, the burst adapter issues the maximum sized bursts with appropriate byte enables. This brings the burst-in-progress up to an aligned slave address. Then, it completes the burst on aligned addresses.

The burst adapter supports variable wrap or sequential burst types to accommodate different properties of memory-mapped masters. Some bursting masters can issue more than one burst type.

Burst adaptation is available for Avalon to Avalon, Avalon to AXI, and AXI to Avalon, and AXI to AXI connections. For information about AXI-to-AXI adaptation, refer to *AXI Wide-to-Narrow Adaptation*

Note: For AXI4 to AXI3 connections, Qsys Pro follows an AXI4 256 burst length to AXI3 16 burst length.

11.1.9.1 Burst Adapter Implementation Options

Qsys Pro automatically inserts burst adapters into your system depending on your master and slave connections, and properties. You can select burst adapter implementation options on the **Interconnect Requirements** tab.

To access the implementation options, you must select the **Burst adapter implementation** setting for the `$system` identifier.

- **Generic converter (slower, lower area)**—Default. Controls all burst conversions with a single converter that is able to adapt incoming burst types. This results in an adapter that has lower f_{MAX}, but smaller area.
- **Per-burst-type converter (faster, higher area)**—Controls incoming bursts with a particular converter, depending on the burst type. This results in an adapter that has higher f_{MAX}, but higher area. This setting is useful when you have AXI masters or slaves and you want a higher f_{MAX}.

Note: For more information about the **Interconnect Requirements** tab, refer to *Creating a System with Qsys Pro*.

Related Links

Creating a System with Qsys Pro on page 299
Qsys Pro is a system integration tool included as part of the Quartus Prime software.

11.1.9.2 Burst Adaptation: AXI to Avalon

Table 122. Burst Adaptation: AXI to Avalon

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

Burst Type	Behavior
Incrementing	**Sequential Slave** Bursts that exceed `slave_max_burst_length` are converted to multiple sequential bursts of a length less than or equal to the `slave_max_burst_length`. Otherwise, the burst is unconverted. For example, for an Avalon slave with a maximum burst length of 4, an `INCR7` burst is converted to `INCR4 + INCR3`. **Wrapping Slave** Bursts that exceed the `slave_max_burst_length` are converted to multiple sequential bursts of length less than or equal to the `slave_max_burst_length`. Bursts that exceed the wrapping boundary are converted to multiple sequential bursts that respect the slave's wrapping boundary.
Wrapping	**Sequential Slave** A WRAP burst is converted to multiple sequential bursts. The sequential bursts are less than or equal to the `max_burst_length` and respect the transaction's wrapping boundary **Wrapping Slave** If the WRAP transaction's boundary matches the slave's boundary, then the burst passes through. Otherwise, the burst is converted to sequential bursts that respect both the transaction and slave wrap boundaries.
Fixed	Fixed bursts are converted to sequential bursts of length 1 that repeatedly access the same address.
Narrow	All narrow-sized bursts are broken into multiple bursts of length 1.

11.1.9.3 Burst Adaptation: Avalon to AXI

Table 123. Burst Adaptation: Avalon to AXI

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

Burst Type	Definition
Sequential	Bursts of length greater than16 are converted to multiple INCR bursts of a length less than or equal to16. Bursts of length less than or equal to16 are not converted.
Wrapping	Only Avalon masters with `alwaysBurstMaxBurst = true` are supported. The WRAP burst is passed through if the length is less than or equal to16. Otherwise, it is converted to two or more INCR bursts that respect the transaction's wrap boundary.
GENERIC_CONVERTER	Controls all burst conversions with a single converter that is able to adapt all incoming burst types. This results in an adapter that has smaller area, but lower f_{MAX}.

11.1.10 Read and Write Responses

Qsys Pro merges write responses if a write is converted (burst adapted) into multiple bursts. Qsys Pro requires read response merging for a downsized (wide-to-narrow width adapted) read.

Qsys Pro merges responses based on the following precedence rule:

```
DECERR > SLVERR > OKAY > EXOKAY
```

Adaptation between a master with write responses and a slave without write responses can be costly, especially if there are multiple slaves, or the slave supports bursts. To minimize the cost of logic between slaves, consider placing the slaves that do not have write responses behind a bridge so that the write response adaptation logic cost is only incurred once, at the bridge's slave interface.

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

Figure 192. Mismatched Master and Slave Response Support

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

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

If the bridge does not support responses, then the responses are generated by the interconnect at the slave interface of the bridge, and responses from the endpoint slave are ignored.

For the response case where the transaction violates security settings or uses an illegal address, the interconnect routes the transactions to the default slave. For information about Qsys Pro system security and how to specify a default slave, refer to *Creating a System with Qsys Pro*.

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

Related Links

- Master Network Interfaces on page 634
- Error Correction Coding (ECC) in Qsys Pro Interconnect on page 687
  Error Correction Coding (ECC) allows the Qsys Pro interconnect to detect and correct errors in order to improve data integrity in memory blocks.
- Avalon-MM Interface Signal Roles
- Interface Properties

11.1.11 Qsys Pro Address Decoding

Address decoding logic forwards appropriate addresses to each slave.

Address decoding logic simplifies component design in the following ways:

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

Figure 193. Address Decoding for One Master and Two Slaves

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

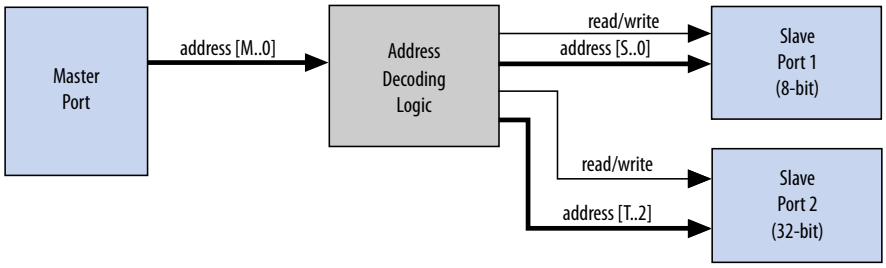
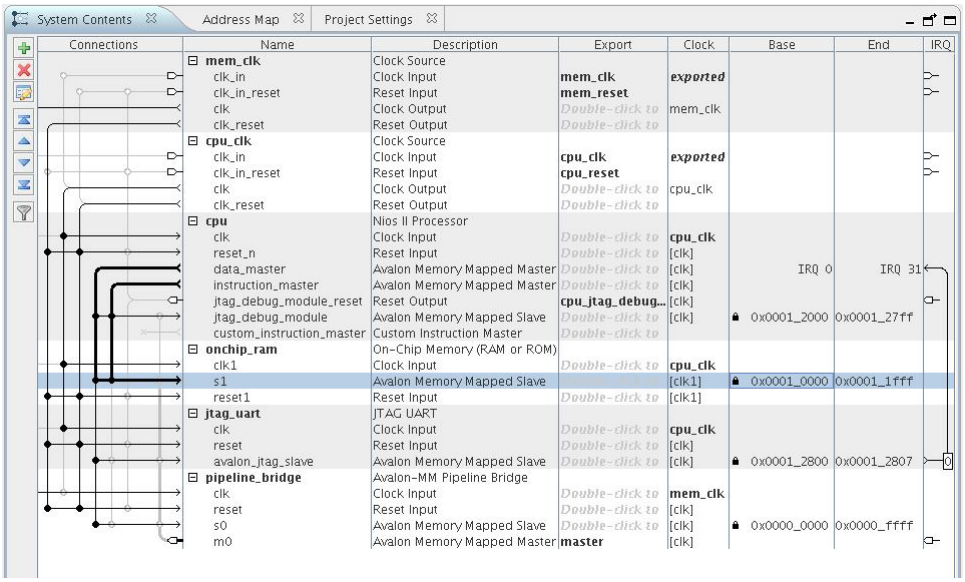


Figure 194. Address Decoding Base Settings

Qsys Pro controls the base addresses with the **Base** setting of active components on the **System Contents** tab. The base address of a slave component must be a multiple of the address span of the component. This restriction is part of the Qsys Pro interconnect to allow the address decoding logic to be efficient, and to achieve the best possible f_{MAX}.



11.2 Avalon Streaming Interfaces

High bandwidth components with streaming data typically use Avalon-ST interfaces for the high throughput datapath. Streaming interfaces can also use memory-mapped connection interfaces to provide an access point for control. In contrast to the memory-mapped interconnect, the Avalon-ST interconnect always creates a point-to-point connection between a single data source and data sink.

Figure 195. Memory-Mapped and Avalon-ST Interfaces

In this example, there are the following connection pairs:

- Data source in the Rx Interface transfers data to the data sink in the FIFO.
- Data source in the FIFO transfers data to the `Tx Interface` data sink.

The memory-mapped interface allows a processor to access the data source, FIFO, or data sink to provide system control. If your source and sink interfaces have different formats, for example, a 32-bit source and an 8-bit sink, Qsys Pro automatically inserts the necessary adapters. You can view the adapters on the **System Contents** tab by clicking **System ➤ Show System with Qsys Pro Interconnect**.

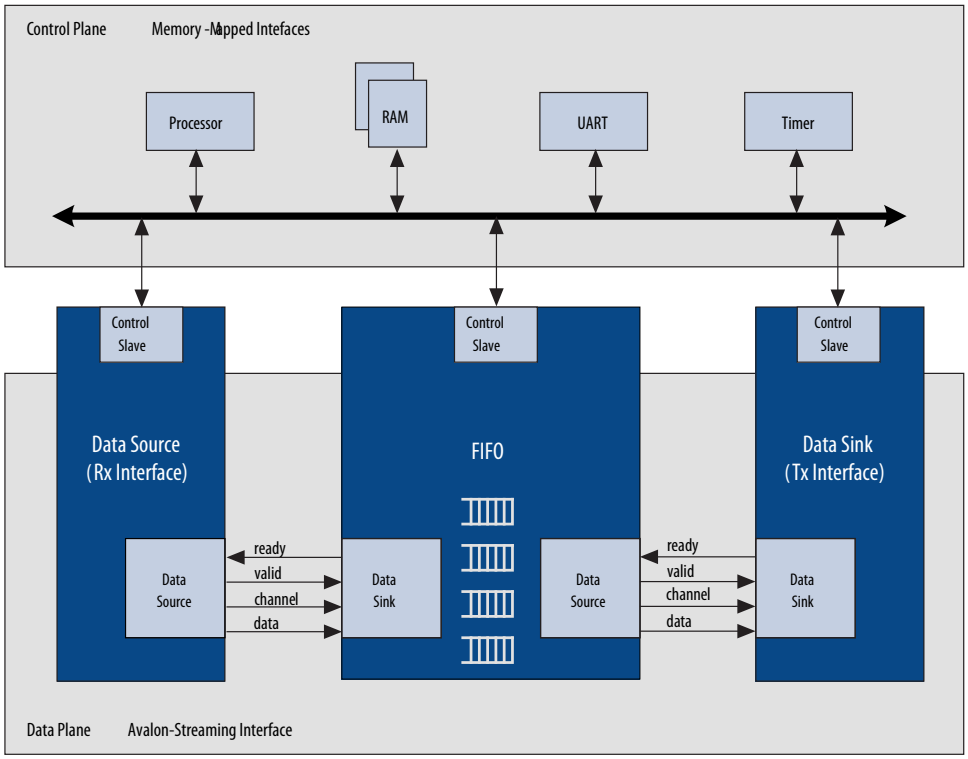


Figure 196. Avalon-ST Connection Between the Source and Sink

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

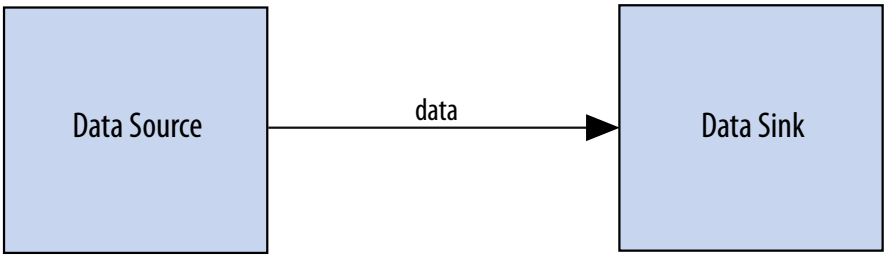
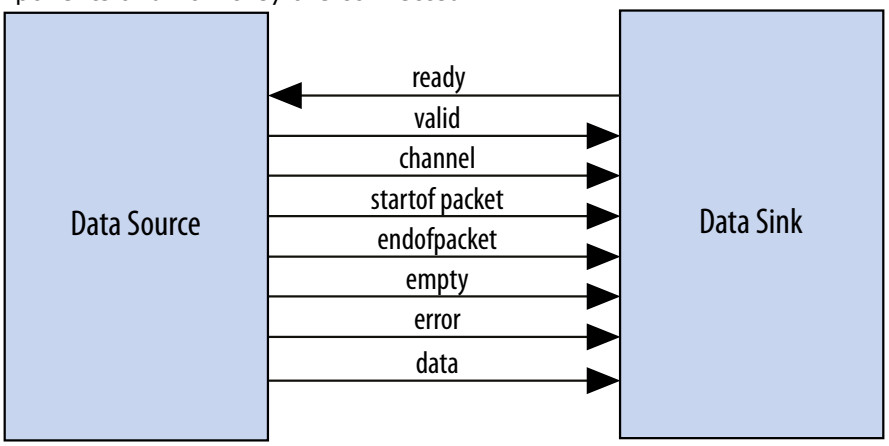


Figure 197. Signals Indicating the Start and End of Packets, Channel Numbers, Error Conditions, and Backpressure

All data transfers using Avalon-ST interconnect occur synchronously on the rising edge of the associated clock interface. Throughput and frequency of a system depends on the components and how they are connected.



The IP Catalog includes a number of Avalon-ST components that you can use to create datapaths, including datapaths whose input and output streams have different properties. Generated systems that include memory-mapped master and slave components may also use these Avalon-ST components because Qsys Pro generation creates interconnect with a structure similar to a network topology, as described in *Qsys Pro Transformations*. The following sections introduce the Avalon-ST components.

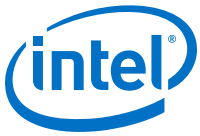
Related Links

- Qsys Pro Transformations on page 632
  The memory-mapped master and slave components connect to network interface modules that encapsulate the transaction in Avalon-ST packets.
- Avalon Interface Specification

11.2.1 Avalon-ST Adapters

Qsys Pro automatically adds Avalon-ST adapters between two components during system generation when it detects mismatched interfaces. If you connect mismatched Avalon-ST sources and sinks, for example, a 32-bit source and an 8-bit sink, Qsys Pro inserts the appropriate adapter type to connect the mismatched interfaces.

After generation, you can view the inserted adapters selecting **System ➤ Show System With Qsys Pro Interconnect**. For each mismatched source-sink pair, Qsys Pro inserts an Avalon-ST Adapter. The adapter instantiates the necessary adaptation logic as sub-components. You can review the logic for each adapter instantiation in the Hierarchy view by expanding each adapter's source and sink interface and comparing the relevant ports. For example, to determine why a channel adapter is inserted, expand the channel adapter's sink and source interfaces and review the channel port properties for each interface.



You can turn off the auto-inserted adapters feature by adding the `qsys_enable_avalon_streaming_transform=off` command to the `quartus.ini` file. When you turn off the auto-inserted adapters feature, if mismatched interfaces are detected during system generation, Qsys Pro does not insert adapters and reports the mismatched interface with validation error message.

Note: The auto-inserted adapters feature does not work for video IP core connections.

11.2.1.1 Avalon-ST Adapter

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

11.2.1.1.1 Avalon-ST Adapter Parameters Common to Source and Sink Interfaces

Table 124. Avalon-ST Adapter Parameters Common to Source and Sink Interfaces

Parameter Name	Description
Symbol Width	Width of a single symbol in bits.
Use Packet	Indicates whether the source and sink interfaces connected to the adapter's source and sink interfaces include the `startofpacket` and `endofpacket` signals, and the optional `empty` signal.

11.2.1.1.2 Avalon-ST Adapter Upstream Source Interface Parameters

Table 125. Avalon-ST Adapter Upstream Source Interface Parameters

Parameter Name	Description
Source Data Width	Controls the data width of the source interface `data` port.
Source Top Channel	Maximum number of output channels allowed.
Source Channel Port Width	Sets the bit width of the source interface `channel` port. If set to 0, there is no `channel` port on the sink interface.
Source Error Port Width	Sets the bit width of the source interface `error` port. If set to 0, there is no `error` port on the sink interface.
Source Error Descriptors	A list of strings that describe the error conditions for each bit of the source interface `error` signal.
Source Uses Empty Port	Indicates whether the source interface includes the `empty` port, and whether the sink interface should also include the `empty` port.
Source Empty Port Width	Indicates the bit width of the source interface `empty` port, and sets the bit width of the sink interface `empty` port.
Source Uses Valid Port	Indicates whether the source interface connected to the sink interface uses the `valid` port, and if set, configures the sink interface to use the `valid` port.
Source Uses Ready Port	Indicates whether the sink interface uses the `ready` port, and if set, configures the source interface to use the `ready` port.
Source Ready Latency	Specifies what ready latency to expect from the source interface connected to the adapter's sink interface.

11.2.1.1.3 Avalon-ST Adapter Downstream Sink Interface Parameters

Table 126. Avalon-ST Adapter Downstream Sink Interface Parameters

Parameter Name	Description
Sink Data Width	Indicates the bit width of the `data` port on the sink interface connected to the source interface.
Sink Top Channel	Maximum number of output channels allowed.
Sink Channel Port Width	Indicates the bit width of the `channel` port on the sink interface connected the source interface.
Sink Error Port Width	Indicates the bit width of the `error` port on the sink interface connected to the adapter's source interface. If set to zero, there is no error port on the source interface.
Sink Error Descriptors	A list of strings that describe the error conditions for each bit of the `error` port on the sink interface connected to the source interface.
Sink Uses Empty Port	Indicates whether the sink interface connected to the source interface uses the `empty` port, and whether the source interface should also use the `empty` port.
Sink Empty Port Width	Indicates the bit width of the `empty` port on the sink interface connected to the source interface, and configures a corresponding `empty` port on the source interface.
Sink Uses Valid Port	Indicates whether the sink interface connected to the source interface uses the `valid` port, and if set, configures the source interface to use the `valid` port.
Sink Uses Ready Port	Indicates whether the `ready` port on the sink interface is connected to the source interface , and if set, configures the sink interface to use the ready port.
Sink Ready Latency	Specifies what ready latency to expect from the source interface connected to the sink interface.

11.2.1.2 Channel Adapter

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

Table 127. Channel Adapter Adaptations

Condition	Description of Adapter Logic
The source uses channels, but the sink does not.	Qsys Pro gives a warning at generation time. The adapter provides a simulation error and signals an error for data for any channel from the source other than 0.
The sink has channel, but the source does not.	Qsys Pro gives a warning at generation time, and the channel inputs to the sink are all tied to a logical 0.
The source and sink both support channels, and the source's maximum channel number is less than the sink's maximum channel number.	The source's channel is connected to the sink's channel unchanged. If the sink's channel signal has more bits, the higher bits are tied to a logical 0.
The source and sink both support channels, but the source's maximum channel number is greater than the sink's maximum channel number.	The source's channel is connected to the sink's channel unchanged. If the source's channel signal has more bits, the higher bits are left unconnected. Qsys Pro gives a warning that channel information may be lost. An adapter provides a simulation error message and an error indication if the value of channel from the source is greater than the sink's maximum number of channels. In addition, the `valid` signal to the sink is deasserted so that the sink never sees data for channels that are out of range.

11.2.1.2.1 Avalon-ST Channel Adapter Input Interface Parameters

Table 128. Avalon-ST Channel Adapter Input Interface Parameters

Parameter Name	Description
Channel Signal Width (bits)	Width of the input channel signal in bits
Max Channel	Maximum number of input channels allowed.

11.2.1.2.2 Avalon-ST Channel Adapter Output Interface Parameters

Table 129. Avalon-ST Channel Adapter Output Interface Parameters

Parameter Name	Description
Channel Signal Width (bits)	Width of the output channel signal in bits.
Max Channel	Maximum number of output channels allowed.

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

Table 130. Avalon-ST Channel Adapter Common to Input and Output Interface Parameters

Parameter Name	Description
Data Bits Per Symbol	Number of bits for each symbol in a transfer.
Include Packet Support	When the Avalon-ST Channel adapter supports packets, the `startofpacket`, `endofpacket`, and optional `empty` signals are included on its sink and source interfaces.
Include Empty Signal	Indicates whether an `empty` signal is required.
Data Symbols Per Beat	Number of symbols per transfer.
Support Backpressure with the ready signal	Indicates whether a `ready` signal is required.
Ready Latency	Specifies the ready latency to expect from the sink connected to the module's source interface.
Error Signal Width (bits)	Bit width of the `error` signal.
Error Signal Description	A list of strings that describes what each bit of the `error` signal represents.

11.2.1.3 Data Format Adapter

The data format adapter allows you to connect interfaces that have different values for the parameters defining the `data` signal, or interfaces where the source does not use the `empty` signal, but the sink does use the `empty` signal. One of the most common uses of this adapter is to convert data streams of different widths.

Table 131. Data Format Adapter Adaptations

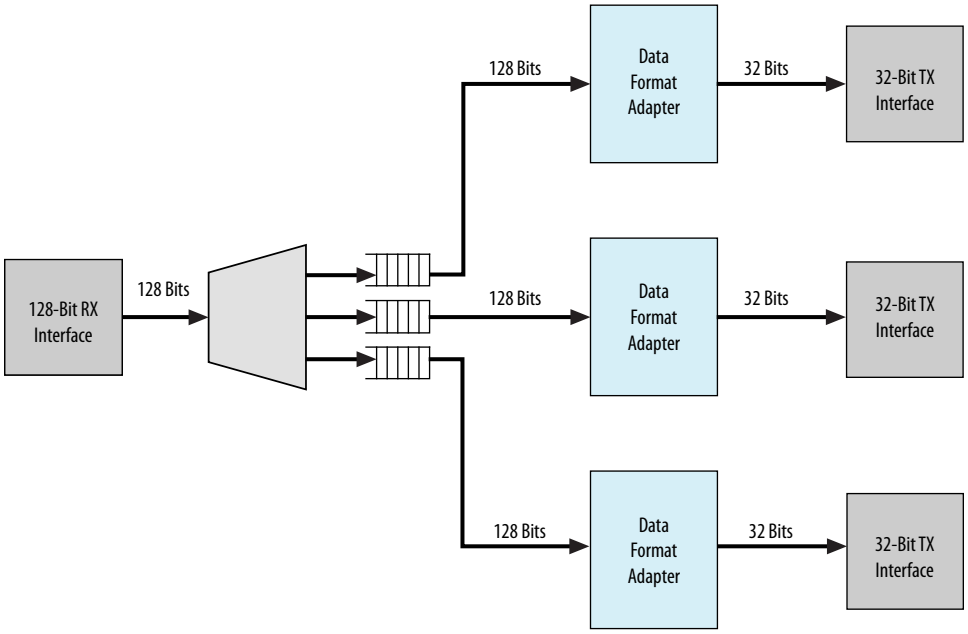
Condition	Description of Adapter Logic
The source and sink's bits per symbol parameters are different.	The connection cannot be made.
The source and sink have a different number of symbols per beat.	The adapter converts the source's width to the sink's width.

continued...

Condition	Description of Adapter Logic
	If the adaptation is from a wider to a narrower interface, a beat of data at the input corresponds to multiple beats of data at the output. If the input `error` signal is asserted for a single beat, it is asserted on output for multiple beats. If the adaptation is from a narrow to a wider interface, multiple input beats are required to fill a single output beat, and the output `error` is the logical OR of the input `error` signal.
The source uses the `empty` signal, but the sink does not use the `empty` signal.	Qsys Pro cannot make the connection.

Figure 198. Avalon Streaming Interconnect with Data Format Adapter

In this example, the data format adapter allows a connection between a 128-bit output data stream and three 32-bit input data streams.



11.2.1.3.1 Avalon-ST Data Format Adapter Input Interface Parameters

Table 132. Avalon-ST Data Format Adapter Input Interface Parameters

Parameter Name	Description
Data Symbols Per Beat	Number of symbols per transfer.
Include Empty Signal	Indicates whether an `empty` signal is required.

11.2.1.3.2 Avalon-ST Data Format Adapter Output Interface Parameters

Table 133. Avalon-ST Data Format Adapter Output Interface Parameters

Parameter Name	Description
Data Symbols Per Beat	Number of symbols per transfer.
Include Empty Signals	Indicates whether an `empty` signal is required.

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

Table 134. Avalon-ST Data Format Adapter Common to Input and Output Interface Parameters

Parameter Name	Description
Data Bits Per Symbol	Number of bits for each symbol in a transfer.
Include Packet Support	When the Avalon-ST Data Format adapter supports packets, Qsys Pro uses `startofpacket`, `endofpacket`, and `empty` signals.
Channel Signal Width (bits)	Width of the output channel signal in bits.
Max Channel	Maximum number of channels allowed.
Read Latency	Specifies the ready latency to expect from the sink connected to the module's source interface.
Error Signal Width (bits)	Width of the `error` signal output in bits.
Error Signal Description	A list of strings that describes what each bit of the `error` signal represents.

11.2.1.4 Error Adapter

The error adapter ensures that per-bit-error information provided by the source interface is correctly connected to the sink interface's input error signal. Error conditions that both the source and sink are able to process are connected. If the source has an `error` signal representing an error condition that is not supported by the sink, the signal is left unconnected; the adapter provides a simulation error message and an error indication if the error is asserted. If the sink has an error condition that is not supported by the source, the sink's input error bit corresponding to that condition is set to 0.

Note: The output interface error signal descriptor accepts an error set with an `other` descriptor. Qsys Pro assigns the bit-wise `ORing` of all input error bits that are unmatched, to the output interface error bits set with the `other` descriptor.

11.2.1.4.1 Avalon-ST Error Adapter Input Interface Parameters

Table 135. Avalon-ST Error Adapter Input Interface Parameters

Parameter Name	Description
Error Signal Width (bits)	The width of the `error` signal. Valid values are 0–256 bits. Type `0` if the `error` signal is not used.
Error Signal Description	The description for each of the error bits. If scripting, separate the description fields by commas. For a successful connection, the description strings of the error bits in the source and sink must match and are case sensitive.

11.2.1.4.2 Avalon-ST Error Adapter Output Interface Parameters

Table 136. Avalon-ST Error Adapter Output Interface Parameters

Parameter Name	Description
Error Signal Width (bits)	The width of the `error` signal. Valid values are 0–256 bits. Type `0` if you do not need to send error values.
Error Signal Description	The description for each of the error bits. Separate the description fields by commas. For successful connection, the description of the error bits in the source and sink must match, and are case sensitive.

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

Table 137. Avalon-ST Error Adapter Common to Input and Output Interface Parameters

Parameter Name	Description
Support Backpressure with the ready signal	Turn on this option to add the backpressure functionality to the interface.
Ready Latency	When the `ready` signal is used, the value for `ready_latency` indicates the number of cycles between when the ready signal is asserted and when valid data is driven.
Channel Signal Width (bits)	The width of the `channel` signal. A channel width of 4 allows up to 16 channels. The maximum width of the `channel` signal is eight bits. Set to 0 if channels are not used.
Max Channel	The maximum number of channels that the interface supports. Valid values are 0–255.
Data Bits Per Symbol	Number of bits per symbol.
Data Symbols Per Beat	Number of symbols per active transfer.
Include Packet Support	Turn on this option if the connected interfaces support a packet protocol, including the `startofpacket`, `endofpacket` and `empty` signals.
Include Empty Signal	Turn this option on if the cycle that includes the `endofpacket` signal can include empty symbols. This signal is not necessary if the number of symbols per beat is 1.

11.2.1.5 Timing Adapter

The timing adapter allows you to connect component interfaces that require a different number of cycles before driving or receiving data. This adapter inserts a FIFO buffer between the source and sink to buffer data or pipeline stages to delay the back pressure signals. You can also use the timing adapter to connect interfaces that support the `ready` signal, and those that do not. The timing adapter treats all signals other than the `ready` and `valid` signals as payload, and simply drives them from the source to the sink.

Table 138. Timing Adapter Adaptations

Condition	Adaptation
The source has `ready`, but the sink does not.	In this case, the source can respond to `backpressure`, but the sink never needs to apply it. The `ready` input to the source interface is connected directly to logical 1.
The source does not have `ready`, but the sink does.	The sink may apply `backpressure`, but the source is unable to respond to it. There is no logic that the adapter can insert that prevents data loss when the source asserts `valid` but the sink is not ready. The adapter provides simulation time error messages if data is lost. The user is presented with a warning, and the connection is allowed.
The source and sink both support backpressure, but the sink's ready latency is greater than the source's.	The source responds to `ready` assertion or deassertion faster than the sink requires it. A number of pipeline stages equal to the difference in ready latency are inserted in the `ready` path from the sink back to the source, causing the source and the sink to see the same cycles as `ready` cycles.
The source and sink both support backpressure, but the sink's ready latency is less than the source's.	The source cannot respond to `ready` assertion or deassertion in time to satisfy the sink. A FIFO whose depth is equal to the difference in ready latency is inserted to compensate for the source's inability to respond in time.

11.2.1.5.1 Avalon-ST Timing Adapter Input Interface Parameters

Table 139. Avalon-ST Timing Adapter Input Interface Parameters

Parameter Name	Description
Support Backpressure with the ready signal	Indicates whether a `ready` signal is required.
Read Latency	Specifies the ready latency to expect from the sink connected to the module's source interface.
Include Valid Signal	Indicates whether the sink interface requires a valid signal.

11.2.1.5.2 Avalon-ST Timing Adapter Output Interface Parameters

Table 140. Avalon-ST Timing Adapter Output Interface Parameters

Parameter Name	Description
Support Backpressure with the ready signal	Indicates whether a `ready` signal is required.
Read Latency	Specifies the ready latency to expect from the sink connected to the module's source interface.
Include Valid Signal	Indicates whether the sink interface requires a valid signal.

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

Table 141. Avalon-ST Timing Adapter Common to Input and Output Interface Parameters

Parameter Name	Description
Data Bits Per Symbol	Number of bits for each symbol in a transfer.
Include Packet Support	Turn this option on if the connected interfaces support a packet protocol, including the `startofpacket`, `endofpacket` and `empty` signals.
Include Empty Signal	Turn this option on if the cycle that includes the `endofpacket` signal can include empty symbols. This signal is not necessary if the number of symbols per beat is 1.
Data Symbols Per Beat	Number of symbols per active transfer.

continued...

Parameter Name	Description
Channel Signal Width (bits)	Width of the output channel signal in bits.
Max Channel	Maximum number of output channels allowed.
Error Signal Width (bits)	Width of the output `error` signal in bits.
Error Signal Description	A list of strings that describes errors.

11.3 Interrupt Interfaces

Using individual requests, the interrupt logic can process up to 32 IRQ inputs connected to each interrupt receiver. With this logic, the interrupt sender connected to interrupt `receiver_0` is the highest priority with sequential receivers being successively lower priority. You can redefine the priority of interrupt senders by instantiating the IRQ mapper component. For more information refer to *IRQ Mapper*.

You can define the interrupt sender interface as asynchronous with no associated clock or reset interfaces. You can also define the interrupt receiver interface as asynchronous with no associated clock or reset interfaces. As a result, the receiver does its own synchronization internally. Qsys Pro does not insert interrupt synchronizers for such receivers.

For clock crossing adaption on interrupts, Qsys Pro inserts a synchronizer, which is clocked with the interrupt end point interface clock when the corresponding starting point interrupt interface has no clock or a different clock (than the end point). Qsys Pro inserts the adapter if there is any kind of mismatch between the start and end points. Qsys Pro does not insert the adapter if the interrupt receiver does not have an associated clock.

Related Links

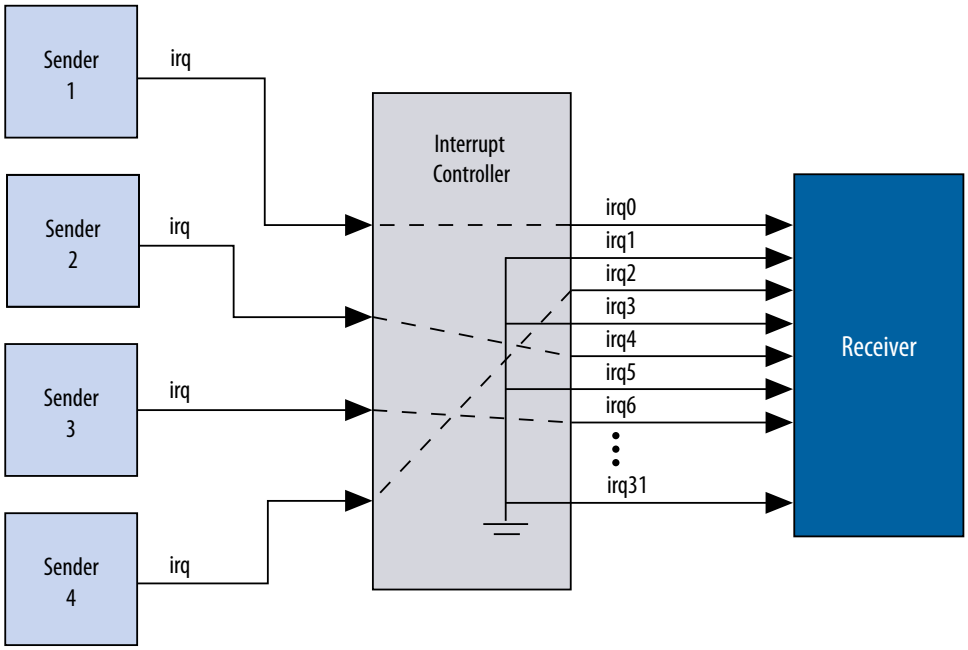
IRQ Mapper on page 664
Qsys Pro inserts the IRQ Mapper automatically during generation. The IRQ Mapper converts individual interrupt wires to a bus, and then maps the appropriate IRQ priority number onto the bus.

11.3.1 Individual Requests IRQ Scheme

In the individual requests IRQ scheme, Qsys Pro interconnect passes IRQs directly from the sender to the receiver, without making assumptions about IRQ priority. In the event that multiple senders assert their IRQs simultaneously, the receiver logic determines which IRQ has highest priority, and then responds appropriately.

Figure 199. Interrupt Controller Mapping IRQs

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



11.3.2 Assigning IRQs in Qsys Pro

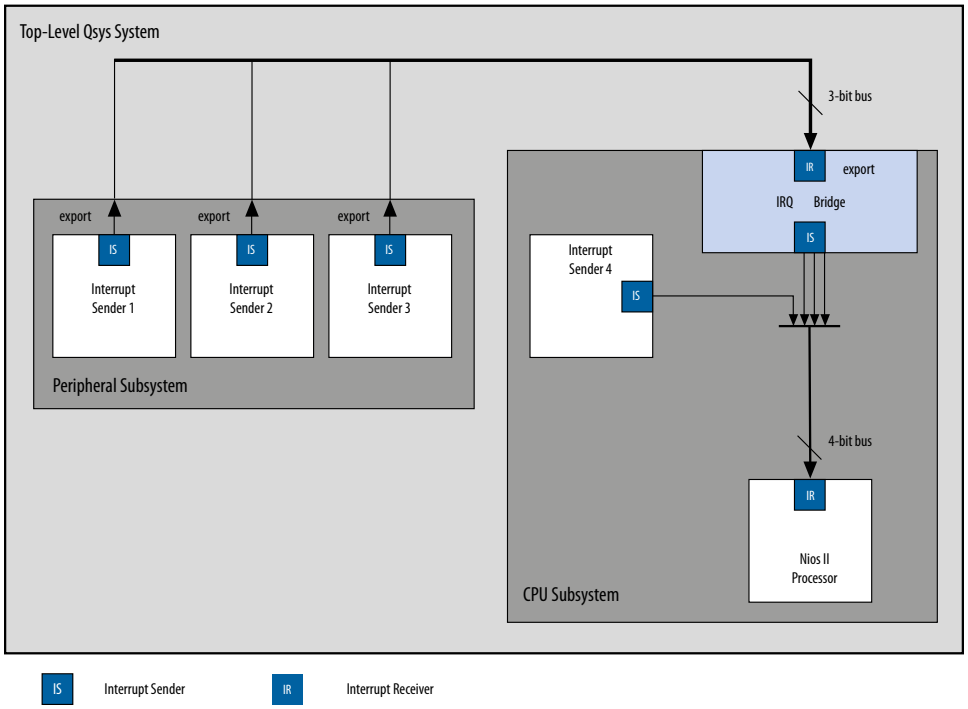
You assign IRQ connections on the **System Contents** tab of Qsys Pro. After adding all components to the system, you connect interrupt senders and receivers. You can use the **IRQ** column to specify an IRQ number with respect to each receiver, or to specify a receiver's IRQ as unconnected. Qsys Pro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

11.3.2.1 IRQ Bridge

The IRQ Bridge allows you to route interrupt wires between Qsys Pro subsystems.

Figure 200. Qsys Pro IRQ Bridge Application

The peripheral subsystem example below has three interrupt senders that are exported to the to- level of the subsystem. The interrupts are then routed to the CPU subsystem using the IRQ bridge.



Note: Nios II BSP tools support the IRQ Bridge. Interrupts connected via an IRQ Bridge appear in the generated `system.h` file. You can use the following properties with the IRQ Bridge, which do not effect Qsys Pro interconnect generation. Qsys Pro uses these properties to generate the correct IRQ information for downstream tools:

- `set_interface_property` *<sender port>* `bridgesToReceiver` *<receiver port>*— The *<sender port>* of the IP generates a signal that is received on the IP's *<receiver port>*. Sender ports are single bits. Receivers ports can be multiple bits. Qsys Pro requires the `bridgedReceiverOffset` property to identify the *<receiver port>* bit that the *<sender port>* sends.
- `set_interface_property` *<sender port>* `bridgedReceiverOffset` *<port number>*— Indicates the *<port number>* of the receiver port that the *<sender port>* sends.

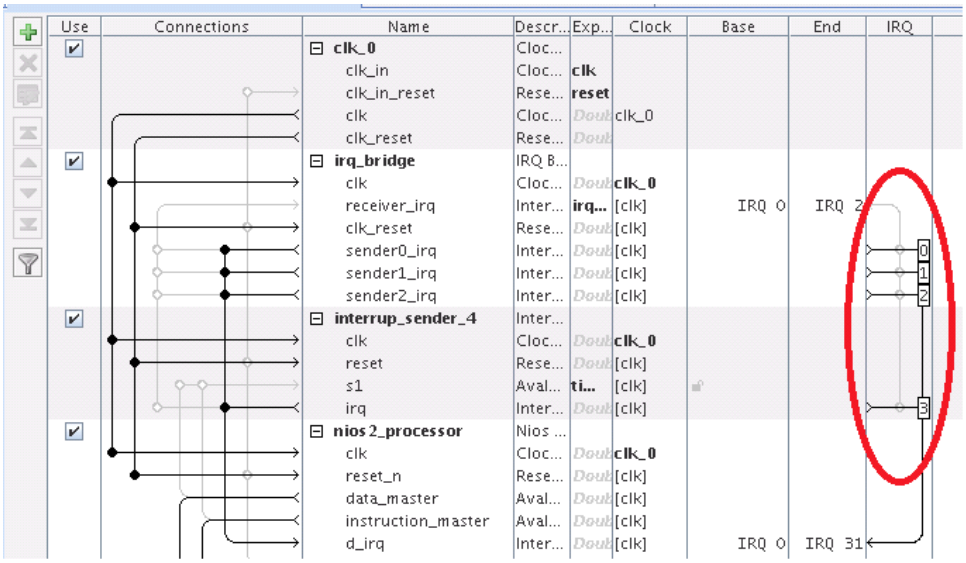
11.3.2.2 IRQ Mapper

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

By default, the interrupt sender connected to the `receiver0` interface of the IRQ mapper is the highest priority, and sequential receivers are successively lower priority. You can modify the interrupt priority of each IRQ wire by modifying the IRQ priority number in Qsys Pro under the **IRQ** column. The modified priority is reflected in the **IRQ_MAP** parameter for the auto-inserted IRQ Mapper.

Figure 201. IRQ Column in Qsys Pro

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



Related Links

IRQ Bridge on page 663
The IRQ Bridge allows you to route interrupt wires between Qsys Pro subsystems.

11.3.2.3 IRQ Clock Crosser

The IRQ Clock Crosser synchronizes interrupt senders and receivers that are in different clock domains. To use this component, connect the clocks for both the interrupt sender and receiver, and for both the interrupt sender and receiver interfaces. Qsys Pro automatically inserts this component when it is required.

11.4 Clock Interfaces

Clock interfaces define the clocks used by a component. Components can have clock inputs, clock outputs, or both. To update the clock frequency of the component, use the **Parameters** tab for the clock source.

The **Clock Source** parameters allows you to set the following options:

- **Clock frequency**—The frequency of the output clock from this clock source.
- **Clock frequency is known**— When turned on, the clock frequency is known. When turned off, the frequency is set from outside the system.

  Note: If turned off, system generation may fail because the components do not receive the necessary clock information. For best results, turn this option on before system generation.
- **Reset synchronous edges**
  - **None**—The reset is asserted and deasserted asynchronously. You can use this setting if you have internal synchronization circuitry that matches the reset required for the IP in the system.
  - **Both**—The reset is asserted and deasserted synchronously.
  - **Deassert**—The reset is deasserted synchronously and asserted asynchronously.

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

Related Links

Recommended Design Practices on page 155
This chapter provides design recommendations for Intel FPGA devices.

11.4.1 (High Speed Serial Interface) HSSI Clock Interfaces

You can use HSSI Serial Clock and HSSI Bonded Clock interfaces in Qsys Pro to enable high speed serial connectivity between clocks that are used by certain IP protocols.

11.4.1.1 HSSI Serial Clock Interface

You can connect the HSSI Serial Clock interface with only similar type of interfaces, for example, you can connect a HSSI Serial Clock Source interface to a HSSI Serial Clock Sink interface.

11.4.1.1.1 HSSI Serial Clock Source

The HSSI Serial Clock interface includes a source in the **Start** direction.

You can instantiate the HSSI Serial Clock Source interface in the **_hw.tcl** file as:

```
add_interface <name> hssi_serial_clock start
```

You can connect the HSSI Serial Clock Source to multiple HSSI Serial Clock Sinks because the HSSI Serial Clock Source supports multiple fan-outs. This Interface has a single **clk** port role limited to a 1 bit width, and a **clockRate** parameter, which is the frequency of the clock driven by the HSSI Serial Clock Source interface.

An unconnected and unexported HSSI Serial Source is valid and does not generate error messages.

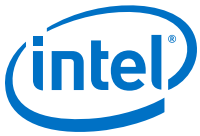


Table 142. HSSI Serial Clock Source Port Roles

Name	Direction	Width	Description
clk	Output	1 bit	A single bit wide port role, which provides synchronization for internal logic.

Table 143. HSSI Serial Clock Source Parameters

Name	Type	Default	Derived	Description
clockRate	long	0	No	The frequency of the clock driven byte HSSI Serial Clock Source interface.

11.4.1.1.2 HSSI Serial Clock Sink

The HSSI Serial Clock interface includes a sink in the **End** direction.

You can instantiate the HSSI Serial Clock Sink interface in the **_hw.tcl** file as:

```
add_interface <name> hssi_serial_clock end
```

You can connect the HSSI Serial Clock Sink interface to a single HSSI Serial Clock Source interface; you cannot connect it to multiple sources. This Interface has a single **clk** port role limited to a 1 bit width, and a **clockRate** parameter, which is the frequency of the clock driven by the HSSI Serial Clock Source interface.

An unconnected and unexported HSSI Serial Sink is invalid and generates error messages.

Table 144. HSSI Serial Clock Sink Port Roles

Name	Direction	Width	Description
clk	Output	1	A single bit wide port role, which provides synchronization for internal logic

Table 145. HSSI Serial Clock Sink Parameters

Name	Type	Default	Derived	Description
clockRate	long	0	No	The frequency of the clock driven by the HSSI Serial Clock Source interface. When you specify a **clockRate** greater than 0, then this interface can be driven only at that rate.

11.4.1.1.3 HSSI Serial Clock Connection

The HSSI Serial Clock Connection defines a connection between a HSSI Serial Clock Source connection point, and a HSSI Serial Clock Sink connection point.

A valid HSSI Serial Clock Connection exists when all of the following criteria are satisfied. If the following criteria are not satisfied, Qsys Pro generates error messages and the connection is prohibited.

- The starting connection point is an HSSI Serial Clock Source with a single port role **clk** and maximum 1 bit in width. The direction of the starting port is **Output**.
- The ending connection point is an HSSI Serial Clock Sink with a single port role **clk**, and maximum 1 bit in width. The direction of the ending port is **Input**.
- If the parameter, **clockRate** of the HSSI Serial Clock Sink is greater than 0, the connection is only valid if the **clockRate** of the HSSI Serial Clock Source is the same as the **clockRate** of the HSSI Serial Clock Sink.

11.4.1.1.4 HSSI Serial Clock Example

Example 100. HSSI Serial Clock Interface Example

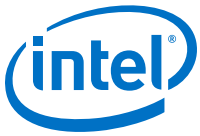
You can make connections to declare the HSSI Serial Clock interfaces in the **_hw.tcl**.

```
package require -exact qsys 14.0

set_module_property name hssi_serial_component
set_module_property ELABORATION_CALLBACK elaborate

add_fileset QUARTUS_SYNTH QUARTUS_SYNTH generate
add_fileset SIM_VERILOG SIM_VERILOG generate
add_fileset SIM_VHDL SIM_VHDL generate

set_fileset_property QUARTUS_SYNTH TOP_LEVEL \
"hssi_serial_component"

set_fileset_property SIM_VERILOG TOP_LEVEL "hssi_serial_component"
set_fileset_property SIM_VHDL TOP_LEVEL "hssi_serial_component"

proc elaborate {} {
    # declaring HSSI Serial Clock Source
    add_interface my_clock_start hssi_serial_clock start
    set_interface_property my_clock_start  ENABLED true

    add_interface_port my_clock_start  hssi_serial_clock_port_out \
 clk Output 1

    # declaring HSSI Serial Clock Sink
    add_interface my_clock_end hssi_serial_clock end
    set_interface_property my_clock_end  ENABLED true

    add_interface_port my_clock_end  hssi_serial_clock_port_in clk \
 Input 1
}

proc generate { output_name } {

    add_fileset_file hssi_serial_component.v VERILOG PATH \
 "hssi_serial_component.v"
}
```

Example 101. HSSI Serial Clock Instantiated in a Composed Component

If you use the components in a hierarchy, for example, instantiated in a composed component, you can declare the connections as illustrated in this example.

```
add_instance myinst1 hssi_serial_component
add_instance myinst2 hssi_serial_component
# add connection from source of myinst1 to sink of myinst2

add_connection myinst1.my_clock_start myinst2.my_clock_end \
hssi_serial_clock

# adding connection from source of myinst2 to sink of myinst1

add_connection myinst2.my_clock_start myinst2.my_clock_end \
hssi_serial_clock
```



11.4.1.2 HSSI Bonded Clock Interface

You can connect the HSSI Bonded Clock interface with only similar type of Interfaces, for example, you can connect a HSSI Bonded Clock Source interface to a HSSI Bonded Clock Sink interface.

11.4.1.2.1 HSSI Bonded Clock Source

The HSSI Bonded Clock interface includes a source in the **Start** direction.

You can instantiate the HSSI Bonded Clock Source interface in the **_hw.tcl** file as:

```
add_interface <name> hssi_bonded_clock start
```

You can connect the HSSI Bonded Clock Source to multiple HSSI Bonded Clock Sinks because the HSSI Serial Clock Source supports multiple fanouts. This Interface has a single **clk** port role limited to a width range of 1 to 1024 bits. The HSSI Bonded Clock Source interface has two parameters: **clockRate** and **serialzationFactor**. **clockRate** is the frequency of the clock driven by the HSSI Bonded Clock Source interface, and the **serializationFactor** is the parallel data width that operates the HSSI TX serializer. The serialization factor determines the required frequency and phases of the individual clocks within the HSSI Bonded Clock interface

An unconnected and unexported HSSI Bonded Source is valid and does not generate error messages.

Table 146. HSSI Bonded Clock Source Port Roles

Name	Direction	Width	Description
clk	Output	1 to 24 bits	A multiple bit wide port role which provides synchronization for internal logic.

Table 147. HSSI Bonded Clock Source Parameters

Name	Type	Default	Derived	Description
clockRate	long	0	No	The frequency of the clock driven byte HSSI Serial Clock Source interface.
serializatio n	long	0	No	The serialization factor is the parallel data width that operates the HSSI TX serializer. The serialization factor determines the necessary frequency and phases of the individual clocks within the HSSI Bonded Clock interface.

11.4.1.2.2 HSSI Bonded Clock Sink

The HSSI Bonded Clock interface includes a sink in the **End** direction.

You can instantiate the HSSI Bonded Clock Sink interface in the **_hw.tcl** file as:

```
add_interface <name> hssi_bonded_clock end
```

You can connect the HSSI Bonded Clock Sink interface to a single HSSI Bonded Clock Source interface; you cannot connect it to multiple sources. This Interface has a single **clk** port role limited to a width range of 1 to 1024 bits. The HSSI Bonded Clock Source interface has two parameters: **clockRate** and **serialzationFactor**. **clockRate** is the frequency of the clock driven by the HSSI Bonded Clock Source interface, and the

serialization factor is the parallel data width that operates the HSSI TX serializer. The serialization factor determines the required frequency and phases of the individual clocks within the HSSI Bonded Clock interface

An unconnected and unexported HSSI Bonded Sink is invalid and generates error messages.

Table 148. HSSI Bonded Clock Source Port Roles

Name	Direction	Width	Description
clk	Output	1 to 24 bits	A multiple bit wide port role which provides synchronization for internal logic.

Table 149. HSSI Bonded Clock Source Parameters

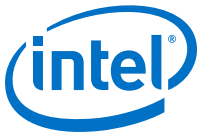
Name	Type	Default	Derived	Description
clockRate	long	0	No	The frequency of the clock driven byte HSSI Serial Clock Source interface.
serialization	long	0	No	The serialization factor is the parallel data width that operates the HSSI TX serializer. The serialization factor determines the necessary frequency and phases of the individual clocks within the HSSI Bonded Clock interface.

11.4.1.2.3 HSSI Bonded Clock Connection

The HSSI Bonded Clock Connection defines a connection between a HSSI Bonded Clock Source connection point, and a HSSI Bonded Clock Sink connection point.

A valid HSSI Bonded Clock Connection exists when all of the following criteria are satisfied. If the following criteria are not satisfied, Qsys Pro generates error messages and the connection is prohibited.

- The starting connection point is an HSSI Bonded Clock Source with a single port role **clk** with a width range of 1 to 24 bits. The direction of the starting port is **Output**.
- The ending connection point is an HSSI Bonded Clock Sink with a single port role **clk** with a width range of 1 to 24 bits. The direction of the ending port is **Input**.
- The width of the starting connection point **clk** must be the same as the width of the ending connection point.
- If the parameter, **clockRate** of the HSSI Bonded Clock Sink greater than 0, then the connection is only valid if the **clockRate** of the HSSI Bonded Clock Source is same as the **clockRate** of the HSSI Bonded Clock Sink.
- If the parameter, **serializationFactor** of the HSSI Bonded Clock Sink is greater than 0, Qsys Pro generates a warning if the **serializationFactor** of HSSI Bonded Clock Source is not same as the **serializationFactor** of the HSSI Bonded Clock Sink.



11.4.1.2.4 HSSI Bonded Clock Example

Example 102. HSSI Bonded Clock Interface Example

You can make connections to declare the HSSI Bonded Clock interfaces in the **_hw.tcl** file.

```
package require -exact qsys 14.0

set_module_property name hssi_bonded_component
set_module_property ELABORATION_CALLBACK elaborate

add_fileset synthesis QUARTUS_SYNTH generate
add_fileset verilog_simulation SIM_VERILOG generate

set_fileset_property synthesis TOP_LEVEL "hssi_bonded_component"

set_fileset_property verilog_simulation TOP_LEVEL \
"hssi_bonded_component"

proc elaborate {} {
    add_interface my_clock_start hssi_bonded_clock start
    set_interface_property my_clock_start  ENABLED true

    add_interface_port my_clock_start  hssi_bonded_clock_port_out \
 clk Output 1024

    add_interface my_clock_end hssi_bonded_clock end
    set_interface_property my_clock_end  ENABLED true

    add_interface_port my_clock_end  hssi_bonded_clock_port_in \
 clk Input 1024
}

proc generate { output_name } {
    add_fileset_file hssi_bonded_component.v VERILOG PATH \
 "hssi_bonded_component.v"}
```

If you use the components in a hierarchy, for example, instantiated in a composed component, you can declare the connections as illustrated in this example.

Example 103. HSII Bonded Clock Instantiated in a Composed Component

```
add_instance myinst1 hssi_bonded_component
add_instance myinst2 hssi_bonded_component
# add connection from source of myinst1 to sink of myinst2

add_connection myinst1.my_clock_start myinst2.my_clock_end \
hssi_bonded_clock

# adding connection from source of myinst2 to sink of myinst1

add_connection myinst2.my_clock_start myinst2.my_clock_end \
hssi_bonded_clock
```

11.5 Reset Interfaces

Reset interfaces provide both soft and hard reset functionality. Soft reset logic typically re-initializes registers and memories without powering down the device. Hard reset logic initializes the device after power-on. You can define separate reset sources for each clock domain, a single reset source for all clocks, or any combination in between.

You can choose to create a single global reset domain by selecting **Create Global Reset Network** on the System menu. If your design requires more than one reset domain, you can implement your own reset logic and connectivity. The IP Catalog includes a reset controller, reset sequencer, and a reset bridge to implement the reset functionality. You can also design your own reset logic.

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

11.5.1 Single Global Reset Signal Implemented by Qsys Pro

If you select **Create Global Reset Network** on the System menu, the Qsys Pro interconnect creates a global reset bus. All of the reset requests are ORed together, synchronized to each clock domain, and fed to the reset inputs. The duration of the reset signal is at least one clock period.

The Qsys Pro interconnect inserts the system-wide reset under the following conditions:

- The global reset input to the Qsys Pro system is asserted.
- Any component asserts its `resetrequest` signal.

11.5.2 Reset Controller

Qsys Pro automatically inserts a reset controller block if the input reset source does not have a reset request, but the connected reset sink requires a reset request.

The Reset Controller has the following parameters that you can specify to customize its behavior:

- **Number of inputs**— Indicates the number of individual reset interfaces the controller ORs to create a signal reset output.
- **Output reset synchronous edges**—Specifies the level of synchronization. You can select one the following options:
  - **None**—The reset is asserted and deasserted asynchronously. You can use this setting if you have designed internal synchronization circuitry that matches the reset style required for the IP in the system.
  - **Both**—The reset is asserted and deasserted synchronously.
  - **Deassert**—The reset is deasserted synchronously and asserted asynchronously.
- **Synchronization depth**—Specifies the number of register stages the synchronizer uses to eliminate the propagation of metastable events.
- **Reset request**—Enables reset request generation, which is an early signal that is asserted before reset assertion. The reset request is used by blocks that require protection from asynchronous inputs, for example, M20K blocks.

Qsys Pro automatically inserts reset synchronizers under the following conditions:

- More than one reset source is connected to a reset sink
- There is a mismatch between the reset source's synchronous edges and the reset sinks' synchronous edges

11.5.3 Reset Bridge

The Reset Bridge allows you to use a reset signal in two or more subsystems of your Qsys Pro system. You can connect one reset source to local components, and export one or more to other subsystems, as required.

The Reset Bridge parameters are used to describe the incoming reset and include the following options:

- **Active low reset**—When turned on, reset is asserted low.
- **Synchronous edges**—Specifies the level of synchronization and includes the following options:
  - **None**—The reset is asserted and deasserted asynchronously. Use this setting if you have internal synchronization circuitry.
  - **Both**—The reset is asserted and deasserted synchronously.
  - **Deassert**—The reset is deasserted synchronously, and asserted asynchronously.
- **Number of reset outputs**—The number of reset interfaces that are exported.

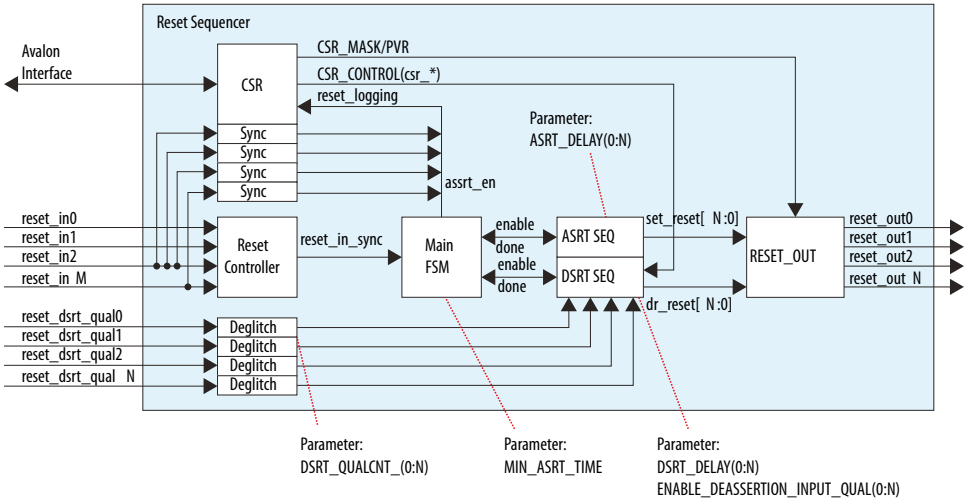
Note: Qsys Pro supports multiple reset sink connections to a single reset source interface. However, there are situations in composed systems where an internally generated reset must be exported from the composed system in addition to being used to connect internal components. In this situation, you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

11.5.4 Reset Sequencer

The Reset Sequencer allows you to control the assertion and deassertion sequence for Qsys Pro system resets.

The Parameter Editor displays the expected assertion and deassertion sequences based on the current settings. You can connect multiple reset sources to the reset sequencer, and then connect the output of the reset sequencer to components in the system.

Figure 202. Elements and Flow of a Reset Sequencer



- Reset Controller—Reused reset controller block. It synchronizes the reset inputs into one and feeds into the main FSM of the sequencer block.
- Sync—Synchronization block (double flipflop).
- Deglitch—Deglitch block. This block waits for a signal to be at a level for X clocks before propagating the input to the output.
- CSR—This block contains the CSR Avalon interface and related CSR register and control block in the sequencer.
- Main FSM—Main sequencer. This block determines when assertion/deassertion and assertion hold timing occurs.
- [A/D]SRT SEQ—Generic sequencer block that sequences out assertion/deassertion of reset from 0:N. The block has multiple counters that saturate upon reaching count.
- RESET_OUT—Controls the end output via:
  - Set/clear from the ASRT_SEQ/DSRT_SEQ.
  - Masking/forcing from CSR controls.
  - Remap of numbering (parameterization).

11.5.4.1 Reset Sequencer Parameters

Table 150. Reset Sequencer Parameters

Parameter	Description
Number of reset outputs	Sets the number of output resets to be sequenced, which is the number of output reset signals defined in the component with a range of 2 to 10.
Number of reset inputs	Sets the number of input reset signals to be sequenced, which is the number of input reset signals defined in the component with a range of 1 to 10.
Minimum reset assertion time	Specifies the minimum assertion cycles between the assertion of the last sequenced reset, and the deassertion of the first sequenced reset. The range is 0 to 1023.
Enable Reset Sequencer CSR	Enables CSR functionality of the Reset Sequencer through an Avalon interface.
reset_out#	Lists the reset output signals. Set the parameters in the other columns for each reset signal in the table.

continued...

Parameter	Description
ASRT Seq#	Determines the order of reset assertion. Enter the values 1, 2, 3, etc. to specify the required non-overlapping assertion order. This value determines the ASRT_REMAP value in the component HDL.
ASRT Cycle#	Number of cycles to wait before assertion of the reset. The value set here corresponds to the ASRT_DELAY value in the component HDL.The range is 0 to1023.
DSRT Seq#	Determines the reset order of reset deassertion. Enter the values 1, 2, 3, etc .to specify the required non-overlapping deassertion order. This value determines the DSRT_REMAP value in the component HDL.
DSRT Cycle#/Deglitch#	Number of cycles to wait before deasserting or deglitching the reset. If the **USE_DRST_QUAL** parameter is set to 0, specifies the number of cycles to wait before deasserting the reset. If **USE_DSRT_QUAL** is set to1, specifies the number of cycles to deglitch the input `reset_dsrt_qual` signal. This value determines either the DSRT_DELAY, or the DSRT_QUALCNT value in the component HDL, depending on the **USE_DSRT_QUAL** parameter setting. The range is 0 to 1023.
USE_DSRT_QUAL	If you set **USE_DSRT_QUAL** to 1, the deassertion sequence waits for an external input signal for sequence qualification instead of waiting for a fixed delay count. To use a fixed delay count for deassertion, set this parameter to 0.

11.5.4.2 Reset Sequencer Timing Diagrams

Figure 203. Basic Sequencing

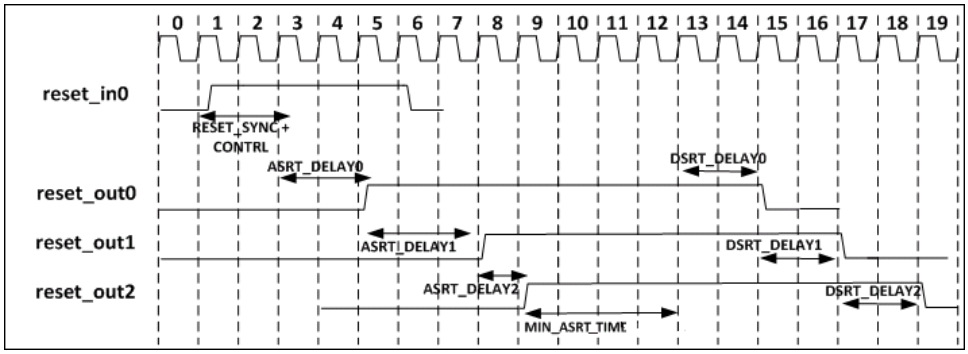
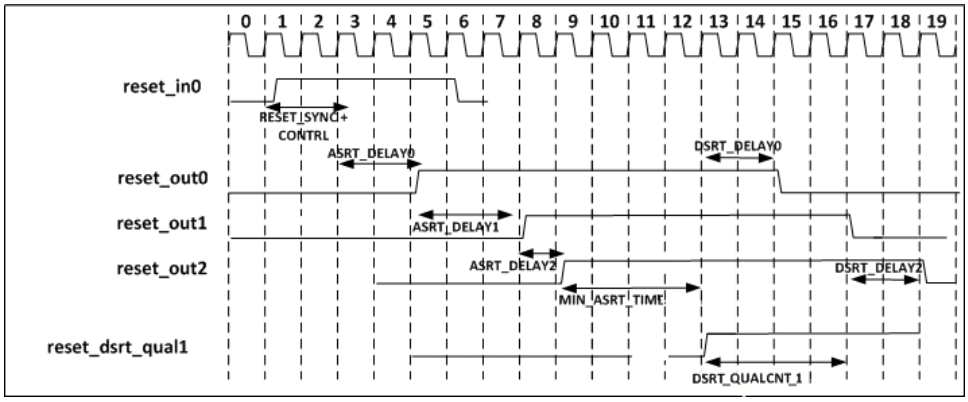


Figure 204. Sequencing with USE_DSRT_QUAL Set



11.5.4.3 Reset Sequencer CSR Registers

The CSR registers on the reset sequencer provide the following functionality:

- **Supports reset logging**
  - Ability to identify which reset is asserted.
  - Ability to determine whether any reset is currently active.
- **Supports software triggered resets**
  - Ability to generate reset by writing to the register.
  - Ability to disable assertion or deassertion sequence.
- **Supports software sequenced reset**
  - Ability for the software to fully control the assertion/deassertion sequence by writing to registers and stepping through the sequence.
- **Support reset override**
  - Ability to assert a particular component reset through software.

11.5.4.3.1 Reset Sequencer Status Register Offset 0x00

The **Status** register contains bits that indicate the sources of resets that cause a reset.

You can clear bits by writing 1 to the bit location. The Reset Sequencer ignores writes to bits with a value of 0. If the sequencer is reset (power-on-reset), all bits are cleared, except the power on reset bit.

Table 151. Values for the Status Register at Offset 0x00

Bit	Attribute	Default	Description
31	RO	0	**Reset Active**—Indicates that the sequencer is currently active in reset sequence (assertion or deassertion).
30	RW1C	0	**Reset Asserted and waiting for SW to proceed:**—Set when there is an active reset assertion, and the next sequence is waiting for the software to proceed. Only valid when the **Enable SW sequenced reset entry** option is turned on.
29	RW1C	0	**Reset Deasserted and waiting for SW to proceed:**—Set when there is an active reset deassertion, and the next sequence is waiting for the software to proceed. Only valid when the **Enable SW sequenced reset bring up** option is turned on.
28:26	RO	0	Reserved.
25:16	RW1C	0	**Reset deassertion input qualification signal reset_dsrt_qual [9:0] status**—Indicates that the reset deassertion's input signal qualification signal is set. This bit is set on the detection of assertion of the signal.
15:12	RO	0	Reserved.
11	RW1C	0	**reset_in9 was triggered**—Indicates that `resetin9` triggered the reset. Cleared by software by writing 1 to this bit location.
10	RW1C	0	**reset_in8 was triggered**—Indicates that `reset_in8` triggered the reset. Cleared by software by writing a1 to this bit location.
9	RW1C	0	**reset_in7 was triggered**—Indicates that `reset_in7` triggered the reset. Cleared by software by writing 1 to this bit location.

continued...

Bit	Attribute	Default	Description
8	RW1C	0	**reset_in6 was triggered**—Indicates that reset_in6 triggered the reset. Cleared by software by writing 1 to this bit location.
7	RW1C	0	**reset_in5 was triggered**—Indicates that reset_in5 triggered the reset. Cleared by software by writing 1 to this bit location.
6	RW1C	0	**reset_in4 was triggered**—Indicates that reset_in4 triggered the reset. Cleared by software by writing 1 to this bit location.
5	RW1C	0	**reset_in3 was triggered**—Indicates that reset_in3 triggered the reset. Cleared by software by writing 1 to this bit location.
4	RW1C	0	**reset_in2 was triggered**—Indicates that reset_in2 triggered the reset. Cleared by software by writing 1 to this bit location.
3	RW1C	0	**reset_in1 was triggered**—Indicates that reset_in1 triggered the reset. Cleared by software by writing 1 to this bit location.
2	RW1C	0	**reset_in0 was triggered**—Indicates that reset_in0 triggered. Cleared by software by writing 1 to this bit location.
1	RW1C	0	**Software triggered reset**—Indicates that the software triggered reset is set by the software, and triggering a reset.
0	RW1C	0	**Power-On-Reset was triggered**—Asserted whenever the reset to the sequencer is triggered. This bit is NOT reset when sequencer is reset. Cleared by software by writing 1 to this bit location.

11.5.4.3.2 Reset Sequencer Interrupt Enable Register Offset 0x04

The Interrupt Enable register contains the interrupt enable bit that you can use to enable any event triggering the IRQ of the reset sequencer.

Table 152. Values for the Interrupt Enable Register at Offset 0x04

Bit	Attribute	Default	Description
31	RO	0	Reserved.
30	RW	0	**Interrupt on Reset Asserted and waiting for SW to proceed** enable. When set, the IRQ is set when the sequencer is waiting for the software to proceed in an assertion sequence.
29	RW	0	**Interrupt on Reset Deasserted and waiting for SW to proceed** enable. When set, the IRQ is set when the sequencer is waiting for the software to proceed in a deassertion sequence.
28:26	RO	0	Reserved.
25:16	RW	0	**Interrupt on Reset deassertion input qualification signal reset_dsrt_qual_[9:0] status**— When set, the IRQ is set when the reset_dsrt_qual[9:0] status bit (per bit enable) is set.
15:12	RO	0	Reserved.
11	RW	0	**Interrupt on reset_in9 Enable**—When set, the IRQ is set when the reset_in9 trigger status bit is set.
10	RW	0	**Interrupt on reset_in8 Enable**—When set, the IRQ is set when the reset_in8 trigger status bit is set.
9	RW	0	**Interrupt on reset_in7 Enable**—When set, the IRQ is set when the reset_in7 trigger status bit is set.
8	RW	0	**Interrupt on reset_in6 Enable**—When set, the IRQ is set when the reset_in6 trigger status bit is set.

continued...

Bit	Attribute	Default	Description
7	RW	0	**Interrupt on reset_in5 Enable**—When set, the IRQ is set when the reset_in5 trigger status bit is set.
6	RW	0	**Interrupt on reset_in4 Enable**—When set, the IRQ is set when the reset_in4 trigger status bit is set.
5	RW	0	**Interrupt on reset_in3 Enable**—When set, the IRQ is set when the reset_in3 trigger status bit is set.
4	RW	0	**Interrupt on reset_in2 Enable**—When set, the IRQ is set when the reset_in2 trigger status bit is set.
3	RW	0	**Interrupt on reset_in1 Enable**—When set, the IRQ is set when the reset_in1 trigger status bit is set.
2	RW	0	**Interrupt on reset_in0 Enable**—When set, the IRQ is set when the reset_in0 trigger status bit is set.
1	RW	0	**Interrupt on Software triggered reset Enable**—When set, the IRQ is set when the software triggered reset status bit is set.
0	RW	0	**Interrupt on Power-On-Reset Enable**—When set, the IRQ is set when the power-on-reset status bit is set.

11.5.4.3.3 Reset Sequencer Control Register Offset 0x08

The Control register contains registers that you can use to control the reset sequencer.

Table 153. Values for the Control Register at Offset 0x08

Bit	Attribute	Default	Description
31:3	RO	0	Reserved.
2	RW	0	**Enable SW sequenced reset entry**—Enable a software sequenced reset entry sequence. Timer delays and input qualification are ignored, and only the software can sequence the entry.
1	RW	0	**Enable SW sequenced reset bring up**—Enable a software sequenced reset bring up sequence. Timer delays and input qualification are ignored, and only the software can sequence the bring up.
0	WO	0	**Initiate Reset Sequence**—Reset Sequencer writes this bit to 1 a single time in order to trigger the hardware sequenced warm reset. Reset Sequencer verifies that **Reset Active** is 0 before setting this bit, and always reads the value 0. To monitor this sequence, verify that **Reset Active** is asserted, and then subsequently deasserted.

11.5.4.3.4 Reset Sequencer Software Sequenced Reset Entry Control Register Offset 0x0C

You can program the Reset Sequencer Software Sequenced Reset Entry Control register to control the reset entry sequence of the sequencer.

When the corresponding enable bit is set, the sequencer stops when the desired reset asserts, and then sets the **Reset Asserted and waiting for SW to proceed** bit. The Reset Sequencer proceeds only after the **Reset Asserted and waiting for SW to proceed** bit is cleared.

Table 154. Values for the Reset Sequencer Software Sequenced Reset Entry Controls Register at Offset 0x0C

Bit	Attribute	Default	Description
31:10	RO	0	Reserved.
9:0	RW	3FF	**Per-reset SW sequenced reset entry enable**—This is a per-bit enable for SW sequenced reset entry. If `bitN` of this register is set, the sequencer sets the `bit30` of the status register when a `resetN` is asserted. It then waits for the `bit30` of the status register to clear before proceeding with the sequence. By default, all bits are enabled (fully SW sequenced).

11.5.4.3.5 Reset Sequencer Software Sequenced Reset Bring Up Control Register Offset 0x10

You can program the Software Sequenced Reset Bring Up Control register to control the reset bring up sequence of the sequencer.

When the corresponding enable bit is set, the sequencer stops when the desired reset asserts, and then sets the **Reset Deasserted and waiting for SW to proceed** bit. The Reset Sequencer proceeds only after the **Reset Deasserted and waiting for SW to proceed** bit is cleared.

Table 155. Values for the Reset Sequencer Software Sequenced Bring Up Control Register at Offset 0x10

Bit	Attribute	Default	Description
31:10	RO	0	Reserved.
9:0	RW	3FF	**Per-reset SW sequenced reset entry enable**—This is a per-bit enable for SW sequenced reset bring up. If `bitN` of this register is set, the sequencer sets `bit29` of the status register when a `resetN` is asserted. It then waits for the `bit29` of the status register to clear before proceeding with the sequence. By default, all bits are enabled (fully SW sequenced).

11.5.4.3.6 Reset Sequencer Software Direct Controlled Resets Offset 0x14

You can write a bit to 1 to assert the `reset_outN` signal, and to 0 to deassert the `reset_outN` signal.

Table 156. Values for the Software Direct Controlled Resets at Offset 0x14

Bit	Attribute	Default	Description
31:26	RO	0	Reserved.
25:16	WO	0	**Reset Overwrite Trigger Enable** —This is a per-bit control trigger bit for the overwrite value to take effect.
15:10	RO	0	Reserved.
9:0	WO	0	**reset_outN Reset Overwrite Value**—This is a per-bit control of the `reset_out` bit. The Reset Sequencer can use this to forcefully drive the reset to a specific value. A value of 1 sets the `reset_out`. A value of 0 clears the `reset_out`. A write to this register only takes effect if the corresponding trigger bit in this register is set.

11.5.4.3.7 Reset Sequencer Software Reset Masking Offset 0x18

You can write a bit to 1 to assert the `reset_outN` signal, and to 0 to deassert the `reset_outN` signal.

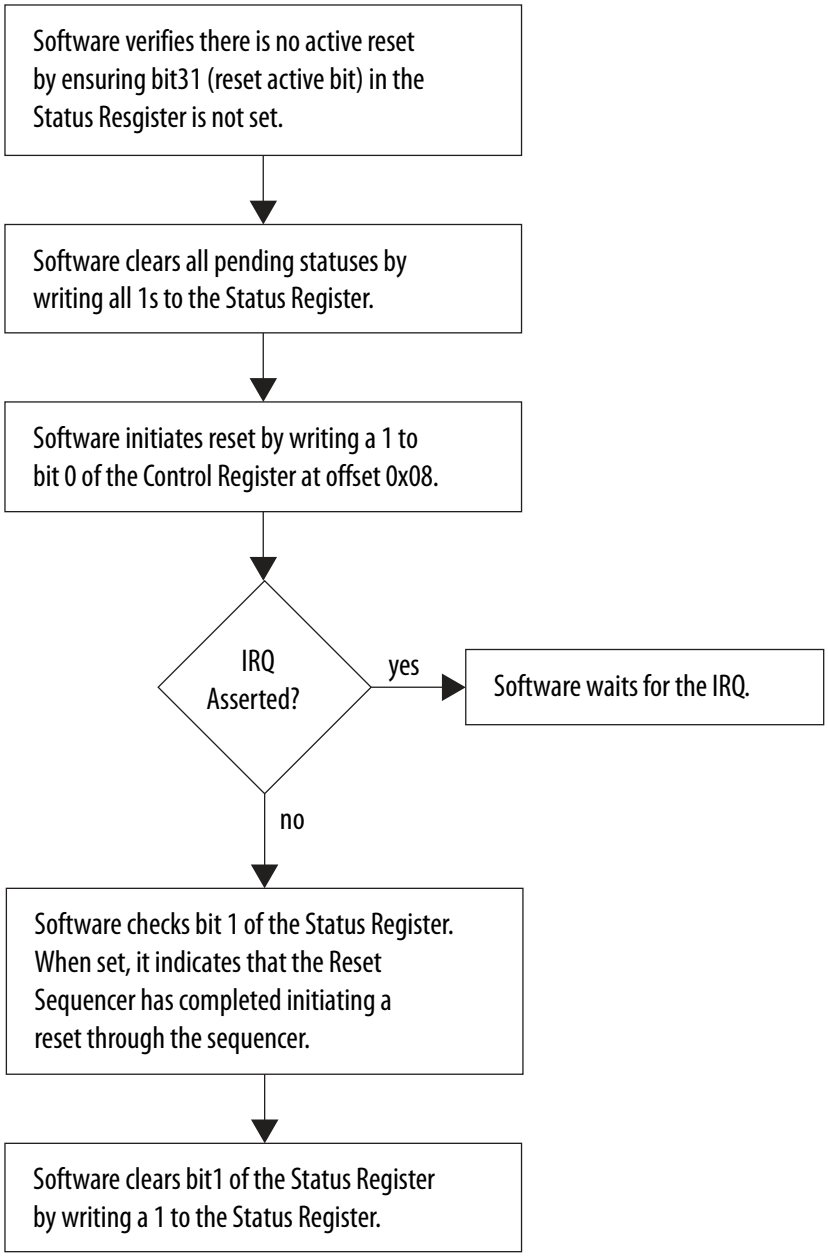
Table 157. Values for the Reset Sequencer Software Reset Masking at Offset 0x18

Bit	Attribute	Default	Description
31:10	RO	0	Reserved.
9:0	RW	0	**reset_outN "Reset Mask Enable"**—This is a per-bit control to mask the `reset_outN` bit. The Software Reset Masking masks the reset bit from being asserted during a reset assertion sequence. If the `reset_out` is already asserted, it does not deassert the reset.

11.5.4.4 Reset Sequencer Software Flows

11.5.4.4.1 Reset Sequencer (Software-Triggered) Flow

Figure 205. Reset Sequencer (Software-Triggered) Flow



11.5.4.4.2 Reset Entry Flow

The following flow sequence occurs for a Reset Entry Flow:

- A reset is triggered either by the software, or when input resets to the Reset Sequencer are asserted.
- The IRQ is asserted, if the IRQ is enabled.
- Software reads the Status register to determine what reset was triggered.

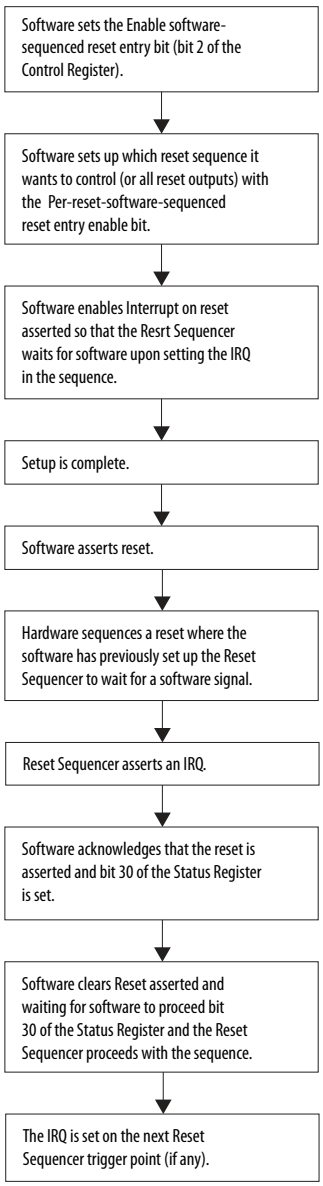
11.5.4.4.3 Reset Bring-Up Flow

The following flow sequence occurs for a Reset Bring-Up Flow:

- When a reset source is deasserted, or when the reset entry sequence has completed without any more pending resets asserted, the bring-up flow is initiated.
- The IRQ is asserted, if the IRQ is enabled.
- Software reads the Status register to determine what reset was triggered.

11.5.4.4.4 Reset Entry (Software-Sequenced) Flow

Figure 206. Reset Entry (Software-Sequenced) Flow



11.5.4.4.5 Reset Bring-Up (Software-Sequenced) Flow

The sequence and flow is similar to the **Reset Entry (SW Sequenced)** flow, though, this flow uses the **reset bring-up** registers/bits in place of the **reset entry** registers/bits.

Related Links

Reset Entry (Software-Sequenced) Flow on page 683

11.6 Conduits

You can use the conduit interface type for interfaces that do not fit any of the other interface types, and to group any arbitrary collection of signals. Like other interface types, you can export or connect conduit interfaces.

The PCI Express-to-Ethernet example in *Creating a System with Qsys Pro* is an example of using a conduit interface for export. You can declare an associated clock interface for conduit interfaces in the same way as memory-mapped interfaces with the `associatedClock`.

To connect two conduit interfaces inside Qsys Pro, the following conditions must be met:

- The interfaces must match exactly with the same signal roles and widths.
- The interfaces must be the opposite directions.
- Clocked conduit connections must have matching `associatedClocks` on each of their endpoint interfaces.

Note: To connect a conduit output to more than one input conduit interface, you can create a custom component. The custom component could have one input that connects to two outputs, and you can use this component between other conduits that you want to connect. For information about the Avalon Conduit interface, refer to the *Avalon Interface Specifications*

Related Links

- Creating a System With Qsys Pro on page 299
  Qsys Pro is a system integration tool included as part of the Quartus Prime software.
- Avalon Interface Specifications

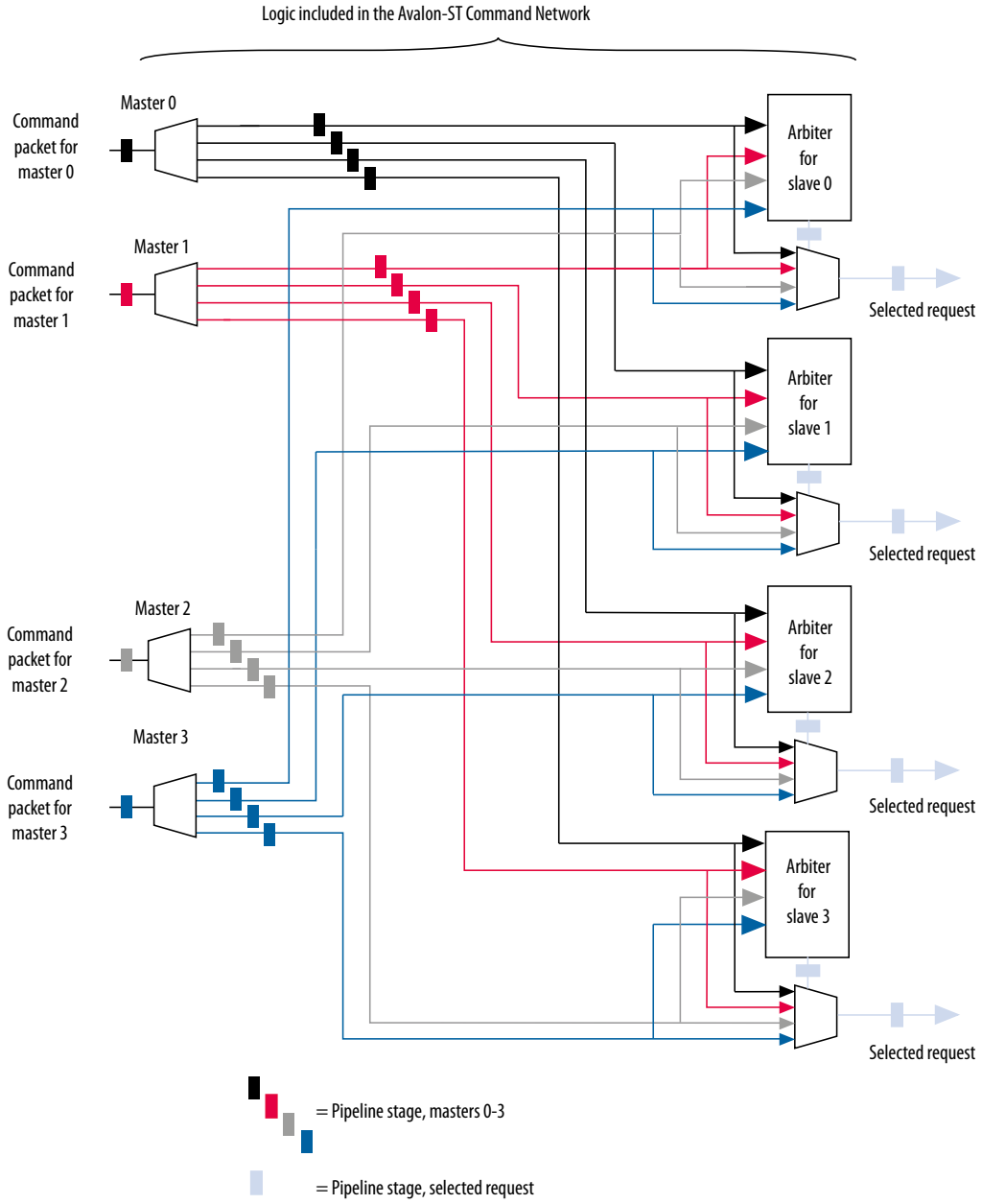
11.7 Interconnect Pipelining

Qsys Pro can automatically insert Avalon-ST pipeline stages when you generate your design.To do so, set the **Limit interconnect pipeline stages to** parameter to a value greater than 0 in the **Project Settings** tab. The pipeline stages increase the f_{MAX} of your design by reducing the combinational logic depth. The cost is additional latency and logic.

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

Figure 207. Pipeline Placement in Arbitration Logic

The example below shows the possible placement of up to four potential pipeline stages, which could be, before the input to the demultiplexer, at the output of the multiplexer, between the arbiter and the multiplexer, and at the outputs of the demultiplexer.



Related Links

- Explore and Manage Qsys Pro Interconnect on page 367

The System with Qsys Pro Interconnect window allows you to see the contents of the Qsys Pro interconnect before you generate your system. In this display of your system, you can review a graphical representation of the generated interconnect. Qsys Pro converts connections between interfaces to interconnect logic during system generation.

- Inserting Pipeline Stages to Increase System Frequency on page 724
  Qsys Pro provides the **Limit interconnect pipeline stages to** option on the **Project Settings** tab to automatically add pipeline stages to the Qsys Pro interconnect when you generate a system.

11.7.1 Manually Controlling Pipelining in the Qsys Pro Interconnect

The **Memory-Mapped Interconnect** tab allows you to manipulate pipleline connections in the Qsys Pro interconnect. Access the **Memory-Mapped Interconnect** tab by clicking **System ➤ Show System With Qsys Pro Interconnect**

Note: To increase interconnect frequency, you should first try increasing the value of the **Limit interconnect pipeline stages to** option on the **Interconnect Requirements** tab. You should only consider manually pipelining the interconnect if changes to this option do not improve frequency, and you have tried all other options to achieve timing closure, including the use of a bridge. Manually pipelining the interconnect should only be applied to complete systems.

1. In the **Interconnect Requirements** tab, first try increasing the value of the **Limit interconnect pipeline stages to** option until it no longer gives significant improvements in frequency, or until it causes unacceptable effects on other parts of the system.
2. In the Quartus Prime software, compile your design and run timing analysis.
3. Using the timing report, identify the critical path through the interconnect and determine the approximate mid-point. The following is an example of a timing report:

```
2.800 0.000 cpu_instruction_master|out_shifter[63]|q
3.004 0.204 mm_domain_0|addr_router_001|Equal5~0|datac
3.246 0.242 mm_domain_0|addr_router_001|Equal5~0|combout
3.346 0.100 mm_domain_0|addr_router_001|Equal5~1|dataa
3.685 0.339 mm_domain_0|addr_router_001|Equal5~1|combout
4.153 0.468 mm_domain_0|addr_router_001|src_channel[5]~0|datad
4.373 0.220 mm_domain_0|addr_router_001|src_channel[5]~0|combout
```

4. In Qsys Pro, click **System ➤ Show System With Qsys Pro Interconnect**.
5. In the **Memory-Mapped Interconnect** tab, select the interconnect module that contains the critical path. You can determine the name of the module from the hierarchical node names in the timing report.
6. Click **Show Pipelinable Locations**. Qsys Pro display all possible pipeline locations in the interconnect. Right-click the possible pipeline location to insert or remove a pipeline stage.

7. Locate the possible pipeline location that is closest to the mid-point of the critical path. The names of the blocks in the memory-mapped interconnect tab correspond to the module instance names in the timing report.
8. Right-click the location where you want to insert a pipeline, and then click **Insert Pipeline**.
9. Regenerate the Qsys Pro system, recompile the design, and then rerun timing analysis. If necessary, repeat the manual pipelining process again until timing requirements are met.

Manual pipelining has the following limitations:

- If you make changes to your original system's connectivity after manually pipelining an interconnect, your inserted pipelines may become invalid. Qsys Pro displays warning messages when you generate your system if invalid pipeline stages are detected. You can remove invalid pipeline stages with the **Remove Stale Pipelines** option in the **Memory-Mapped Interconnect** tab. Intel recommends that you do not make changes to the system's connectivity after manual pipeline insertion.
- Review manually-inserted pipelines when upgrading to newer versions of Qsys Pro. Manually-inserted pipelines in one version of Qsys Pro may not be valid in a future version.

Related Links

- Specify Qsys Pro Interconnect Requirements on page 346
  The **Interconnect Requirements** tab allows you to apply system-wide, `$system`, and interface interconnect requirements for IP components in your system.
- Qsys Pro System Design Components on page 879
  You can use Qsys Pro IP components to create Qsys Pro systems.

11.8 Error Correction Coding (ECC) in Qsys Pro Interconnect

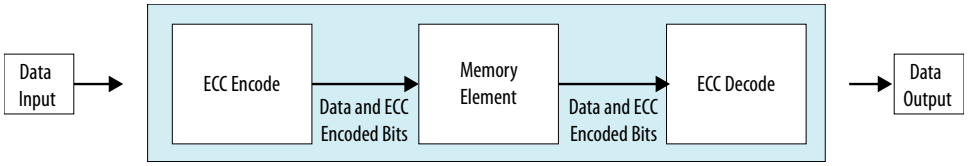
Error Correction Coding (ECC) allows the Qsys Pro interconnect to detect and correct errors in order to improve data integrity in memory blocks.

As transistors become smaller, computer hardware is more susceptible to data corruption. Data corruption causes Single Event Upsets (SEUs) and increases the probability of Failures in Time (FIT) rates in computer systems. SEU events without error notification can cause the system to be stuck in an unknown response state, and increase the probability of FIT rates.

ECC encodes the data bus with a Hamming code before it writes it to the memory device, and then decodes and performs error checking on the data on output.

Note: Qsys Pro sends uncorrectable errors in memeory elements as a DECERR on the response bus. This feature is currently only supported for `rdata_FIFO` instances when back pressure occurs on the `wait_request` signal.

Figure 208. High-Level Implementation of RDATA FIFO with ECC Enabled



Related Links

- Read and Write Responses on page 650
- Specify Qsys Pro Interconnect Requirements

11.9 AMBA 3 AXI Protocol Specification Support (version 1.0)

Qsys Pro allows memory-mapped connections between AXI3 components, AXI3 and AXI4 components, and AXI3 and Avalon interfaces with some unique or exceptional support.

Refer to the *AMBA 3 Protocol Specifications* on the ARM website for more information.

Related Links

AMBA 3 Protocol Specifications

11.9.1 Channels

Qsys Pro has the following support and restrictions for AXI3 channels.

11.9.1.1 Read and Write Address Channels

All signals are allowed with some limitations.

The following limitations are present in Qsys Pro 14.0:

- Supports 64-bit addressing.
- ID width limited to 18-bits.
- HPS-FPGA master interface has a 12-bit ID.

11.9.1.2 Write Data, Write Response, and Read Data Channels

All signals are allowed with some limitations.

The following limitations are present in Qsys Pro 14.0:

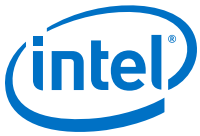
- Data widths limited to a maximum of 1024-bits
- Limited to a fixed byte width of 8-bits

11.9.1.3 Low Power Channel

Low power extensions are not supported in Qsys Pro, version 14.0.

11.9.2 Cache Support

AWCACHE and ARCACHE are passed to an AXI slave unmodified.



11.9.2.1 Bufferable

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

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

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

11.9.2.2 Cacheable (Modifiable)

Qsys Pro interconnect acknowledges the cacheable (modifiable) attribute of AXI transactions.

It does not change the address, burst length, or burst size of non-modifiable transactions, with the following exceptions:

- Qsys Pro considers a wide transaction to a narrow slave as modifiable because the size requires reduction.
- Qsys Pro may consider AXI read and write transactions as modifiable when the destination is an Avalon slave. The AXI transaction may be split into multiple Avalon transactions if the slave is unable to accept the transaction. This may occur because of burst lengths, narrow sizes, or burst types.

Qsys Pro ignores all other bits, for example, read allocate or write allocate because the interconnect does not perform caching. By default, Qsys Pro considers Avalon master transactions as non-bufferable and non-cacheable, with the allocate bits tied low.

11.9.3 Security Support

TrustZone refers to the security extension of the ARM architecture, which includes the concept of "secure" and "non-secure" transactions, and a protocol for processing between the designations.

The interconnect passes the AWPROT and ARPROT signals to the endpoint slave without modification. It does not use or modify the PROT bits.

Refer to *Creating a System with Qsys Pro* for more information about secure systems and the TrustZone feature.

Related Links

Creating a System with Qsys Pro on page 299
Qsys Pro is a system integration tool included as part of the Quartus Prime software.

11.9.4 Atomic Accesses

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

11.9.5 Response Signaling

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

11.9.6 Ordering Model

Qsys Pro interconnect provides responses in the same order as the commands are issued.

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

11.9.6.1 AXI and Avalon Ordering

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

According to the *AMBA Protocol Specifications*, there is no ordering requirement between reads and writes. However, Avalon has an implicit ordering model that requires transactions from a master to the same slave to be in order.

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

11.9.7 Data Buses

Narrow bus transfers are supported. AXI write strobes can have any pattern that is compatible with the address and size information. Intel recommends that transactions to Avalon slaves follow Avalon `byteenable` limitations for maximum compatibility.

Note: Byte `0` is always bits `[7:0]` in the interconnect, following AXI's and Avalon's byte (address) invariance scheme.

11.9.8 Unaligned Address Commands

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

Note: Unaligned transfers are aligned if downsizing occurs. For example, when downsizing to a bus width narrower than that required by the transaction size, AWSIZE or ARSIZE, the transaction must be modified.

11.9.9 Avalon and AXI Transaction Support

Qsys Pro 14.0 supports transactions between Avalon and interfaces with some limitations.

11.9.9.1 Transaction Cannot Cross 4KB Boundaries

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

11.9.9.2 Handling Read Side Effects

Read side effects can occur when more bytes than necessary are read from the slave, and the unwanted data that are read are later inaccessible on subsequent reads. For write commands, the correct byteenable paths are asserted based on the size of the transactions. For read commands, narrow-sized bursts are broken up into multiple non-bursting commands, and each command with the correct byteenable paths asserted.

Qsys Pro always assumes that the byteenable is asserted based on the size of the command, not the address of the command. The following scenarios are examples:

- For a 32-bit AXI master that issues a read command with an unaligned address starting at address `0x01`, and a burstcount of 2 to a 32-bit Avalon slave, the starting address is: `0x00`.
- For a 32-bit AXI master that issues a read command with an unaligned address starting at address `0x01`, with 4-bytes to an 8-bit AXI slave, the starting address is: `0x00`.

11.10 AMBA 3 APB Protocol Specification Support (version 1.0)

AMBA APB provides a low-cost interface that is optimized for minimal power consumption and reduced interface complexity. You can use AMBA APB to interface to peripherals which are low-bandwidth and do not require the high performance of a pipelined bus interface. Signal transitions are sampled at the rising edge of the clock to enable the integration of APB peripherals easily into any design flow.

Qsys Pro allows connections between APB components, and AXI3, AXI4, and Avalon memory-mapped interfaces. The following sections describe unique or exceptional APB support in the Qsys Pro software.

Related Links

AMBA APB Protocol Specifications
SHDS

11.10.1 Bridges

With APB, you cannot use bridge components that use multiple PSELx in Qsys Pro. As a workaround, you can group PSELx, and then send the packet to the slave directly.

Intel recommends as an alternative that you instantiate the APB bridge and all the APB slaves in Qsys Pro. You should then connect the slave side of the bridge to any high speed interface and connect the master side of the bridge to the APB slaves. Qsys Pro creates the interconnect on either side of the APB bridge and creates only one PSEL signal.

Alternatively, you can connect a bridge to the APB bus outside of Qsys Pro. Use an Avalon/AXI bridge to export the Avalon/AXI master to the top-level, and then connect this Avalon/AXI interface to the slave side of the APB bridge. Alternatively, instantiate the APB bridge in Qsys Pro and export APB master to the top- level, and from there connect to APB bus outside of Qsys Pro.

11.10.2 Burst Adaptation

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

11.10.3 Width Adaptation

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

11.10.4 Error Response

Error responses are returned to the master. Qsys Pro performs error mapping if the master is an AXI3 or AXI4 master, for example, RRESP/BRESP= SLVERR. For the case when the slave does not use SLVERR signal, an OKAY response is sent back to master by default.

11.11 AMBA AXI4 Memory-Mapped Interface Support (version 2.0)

Qsys Pro allows memory-mapped connections between AXI4 components, AXI4 and AXI3 components, and AXI4 and Avalon interfaces with some unique or exceptional support.

11.11.1 Burst Support

Qsys Pro supports `INCR` bursts up to 256 beats. Qsys Pro converts long bursts to multiple bursts in a packet with each burst having a length less than or equal to `MAX_BURST` when going to AXI3 or Avalon slaves.

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

Bursts with AXI3 slaves as destinations are shortened to multiple bursts, with each burst length less than or equal to 16. Bursts with AXI4 slaves as destinations are not shortened.

11.11.2 QoS

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

Transactions from AXI3 and Avalon masters have a default value of `4'b0000`, which indicates that the transactions are not part of the QoS flow. QoS values are not used for slaves that do not support QoS.

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

11.11.3 Regions

For Qsys Pro 14.0, there is no support for the optional regions feature. AXI4 slaves with `AXREGION` signals are allowed. `AXREGION` signals are driven with the default value of `0x0`, and are limited to one entry in a master's address map.

11.11.4 Write Response Dependency

Write response dependency as specified in the *AMBA Protocol Specifications* for AXI4 is not supported.

Related Links

AMBA Protocol Specifications

11.11.5 AWCACHE and ARCACHE

For AXI4, Qsys Pro meets the requirement for modifiable and non-modifiable transactions. The modifiable bit refers to `ARCACHE[1]`and `AWCACHE[1]`.

11.11.6 Width Adaptation and Data Packing in Qsys Pro

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

The following rules apply:

- Data packing is supported when masters and slaves are Avalon-MM.
- Data packing is not supported when any master or slave is an AXI3, AXI4, or APB component.

For example, for a read/write command with a 32-bit master connected to a 64-bit slave, and a transaction of 2 burstcounts, Qsys Pro sends 2 separate read/write commands to access the 64-bit data width of the slave. Data packing is only supported if the system does not contain AXI3, AXI4, or APB masters or slaves.

11.11.7 Ordering Model

Out of order support is not implemented in Qsys Pro, version 14.0. Qsys Pro processes AXI slaves as device non-bufferable memory types.

The following describes the required behavior for the device non-bufferable memory type:

- Write response must be obtained from the final destination.
- Read data must be obtained from the final destination.
- Transaction characteristics must not be modified.
- Reads must not be pre-fetched. Writes must not be merged.
- Non-modifiable read and write transactions.

(`AWCACHE[1] = 0 or ARCACHE[1] = 0`) from the same ID to the same slave must remain ordered. The interconnect always provides responses in the same order as the commands issued. Slaves that support reordering provide a constant transaction ID to prevent reordering. AXI slaves that do not reorder are provided with transaction IDs, which allows exclusive accesses to be used for such slaves.

11.11.8 Read and Write Allocate

Read and write allocate does not apply to Qsys Pro interconnect, which does not have caching features, and always receives responses from an endpoint.

11.11.9 Locked Transactions

Locked transactions are not supported for Qsys Pro, version 14.0.

11.11.10 Memory Types

For AXI4, Qsys Pro processes transactions as though the endpoint is a device memory type. For device memory types, using non-bufferable transactions to force previous bufferable transactions to finish is irrelevant, because Qsys Pro interconnect always identifies transactions as being non-bufferable.

11.11.11 Mismatched Attributes

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

11.11.12 Signals

Qsys Pro supports up to 64-bits for the BUSER, WUSER and RUSER sideband signals. AXI4 allows some signals to be omitted from interfaces by aligning them with the default values as defined in the *AMBA Protocol Specifications* on the ARM website.

Related Links

AMBA Protocol Specifications

11.12 AMBA AXI4 Streaming Interface Support (version 1.0)

11.12.1 Connection Points

Qsys Pro allows you to connect an AXI4 stream interface to another AXI4 stream interface.

The connection is point-to-point without adaptation and must be between an `axi4stream_master` and `axi4stream_slave`. Connected interfaces must have the same port roles and widths.

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

11.12.1.1 AXI4 Streaming Connection Point Parameters

Table 158. AXI4 Streaming Connection Point Parameters

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

11.12.1.2 AXI4 Streaming Connection Point Signals

Table 159. AXI4 Stream Connection Point Signals

Port Role	Width	Master Direction	Slave Direction	Required
`tvalid`	1	Output	Input	Yes
`tready`	1	Input	Output	No
`tdata`[5]	8:4096	Output	Input	No
`tstrb`	1:512	Output	Input	No
`tkeep`	1:512	Output	Input	No
`tid`[6]	1:8	Output	Input	No
`tdest`[7]	1:4	Output	Input	No
`tuser`[8]	1:4096	Output	Input	No
`tlast`	1	Output	Input	No

11.12.2 Adaptation

AXI4 stream adaptation support is not available. AXI4 stream master and slave interface signals and widths must match.

5 integer in mutiple of bytes

6 maximum 8-bits

7 maximum 4-bits

8 number of bits in multiple of the number of bytes of `tdata`

11.13 AMBA AXI4-Lite Protocol Specification Support (version 2.0)

AXI4-Lite is a sub-set of AMBA AXI4. It is suitable for simpler control register-style interfaces that do not require the full functionality of AXI4.

Qsys Pro 14.0 supports the following AXI4-Lite features:

- Transactions with a burst length of 1.
- Data accesses use the full width of a data bus (32- bit or 64-bit) for data accesses, and no narrow-size transactions.
- Non-modifiable and non-bufferable accesses.
- No exclusive accesses.

11.13.1 AXI4-Lite Signals

Qsys Pro supports all AXI4-Lite interface signals. All signals are required.

Table 160. AXI4-Lite Signals

Global	Write Address Channel	Write Data Channel	Write Response Channel	Read Address Channel	Read Data Channel
ACLK	AWVALID	WVALID	BVALID	ARVALID	RVALID
ARESETn	AWREADY	WREADY	BREADY	ARREADY	RREADY
-	AWADDR	WDATA	BRESP	ARADDR	RDATA
-	AWPROT	WSTRB	-	ARPROT	RRESP

11.13.2 AXI4-Lite Bus Width

AXI4-Lite masters or slaves must have either 32-bit or 64-bit bus widths. Qsys Pro interconnect inserts a width adapter if a master and slave pair have different widths.

11.13.3 AXI4-Lite Outstanding Transactions

AXI-Lite supports outstanding transactions. The options to control outstanding transactions is set in the parameter editor for the selected component.

11.13.4 AXI4-Lite IDs

AXI4-Lite does not support IDs. Qsys Pro performs ID reflection inside the slave agent.

11.13.5 Connections Between AXI3/4 and AXI4-Lite

11.13.5.1 AXI4-Lite Slave Requirements

For an AXI4-Lite slave side, the master can be any master interface type, such as an Avalon (with bursting), AXI3, or AXI4. Qsys Pro allows the following connections and inserts adapters, if needed.

- **Burst adapter**—Avalon and AXI3 and AXI4 bursting masters require a burst adapter to shorten the burst length to 1 before sending a transaction to an AXI4-Lite slave.
- Qsys Pro interconnect uses a width adapter for mismatched data widths.
- Qsys Pro interconnect performs ID reflection inside the slave agent.
- An AXI4-Lite slave must have an address width of at least 12-bits.
- AXI4-Lite does not have the AXSIZE parameter. Narrow master to a wide AXI4-Lite slave is not supported. For masters that support narrow-sized bursts, for example, AXI3 and AXI4, a burst to an AXI4-Lite slave must have a burst size equal to or greater than the slave's burst size.

11.13.5.2 AXI4-Lite Data Packing

Qsys Pro interconnect does not support AXI4-Lite data packing.

11.13.6 AXI4-Lite Response Merging

When Qsys Pro interconnect merges SLVERR and DECERR, the error responses are not sticky. The response is based on priority and the master always sees a DECERR. When SLVERR and DECERR are merged, it is based on their priorities, not stickiness. DECERR receives priority in this case, even if SLVERR returns first.

11.14 Port Roles (Interface Signal Types)

Each interfaces defines a number of signal roles and their behavior. Many signal roles are optional, allowing IP component designers the flexibility to select only the signal roles necessary to implement the required functionality.

11.14.1 AXI Master Interface Signal Types

Table 161. AXI Master Interface Signal Types

Name	Direction	Width
araddr	output	1 - 64
arburst	output	2
arcache	output	4
arid	output	1 - 18
arlen	output	4
arlock	output	2
arprot	output	3
arready	input	1

continued...

Name	Direction	Width
arsize	output	3
aruser	output	1 - 64
arvalid	output	1
awaddr	output	1 - 64
awburst	output	2
awcache	output	4
awid	output	1 - 18
awlen	output	4
awlock	output	2
awprot	output	3
awready	input	1
awsize	output	3
awuser	output	1 - 64
awvalid	output	1
bid	input	1 - 18
bready	output	1
bresp	input	2
bvalid	input	1
rdata	input	8, 16, 32, 64, 128, 256, 512, 1024
rid	input	1 - 18
rlast	input	1
rready	output	1
rresp	input	2
rvalid	input	1
wdata	output	8, 16, 32, 64, 128, 256, 512, 1024
wid	output	1 - 18
wlast	output	1
wready	input	1
wstrb	output	1, 2, 4, 8, 16, 32, 64, 128
wvalid	output	1

11.14.2 AXI Slave Interface Signal Types

Table 162. AXI Slave Interface Signal Types

Name	Direction	Width
araddr	input	1 - 64
arburst	input	2
arcache	input	4
arid	input	1 - 18
arlen	input	4
arlock	input	2
arprot	input	3
arready	output	1
arsize	input	3
aruser	input	1 - 64
arvalid	input	1
awaddr	input	1 - 64
awburst	input	2
awcache	input	4
awid	input	1 - 18
awlen	input	4
awlock	input	2
awprot	input	3
awready	output	1
awsize	input	3
awuser	input	1 - 64
awvalid	input	1
bid	output	1 - 18
bready	input	1
bresp	output	2
bvalid	output	1
rdata	output	8, 16, 32, 64, 128, 256, 512, 1024
rid	output	1 - 18
rlast	output	1
rready	input	1
rresp	output	2
rvalid	output	1

continued...

Name	Direction	Width
wdata	input	8, 16, 32, 64, 128, 256, 512, 1024
wid	input	1 - 18
wlast	input	1
wready	output	1
wstrb	input	1, 2, 4, 8, 16, 32, 64, 128
wvalid	input	1

11.14.3 AXI4 Master Interface Signal Types

Table 163. AXI4 Master Interface Signal Types

Name	Direction	Width
araddr	output	1 - 64
arburst	output	2
arcache	output	4
arid	output	1 - 18
arlen	output	8
arlock	output	1
arprot	output	3
arready	input	1
arregion	output	1 - 4
arsize	output	3
aruser	output	1 - 64
arvalid	output	1
awaddr	output	1 - 64
awburst	output	2
awcache	output	4
awid	output	1 - 18
awlen	output	8
awlock	output	1
awprot	output	3
awqos	output	1 - 4
awready	input	1
awregion	output	1 - 4
awsize	output	3
awuser	output	1 - 64

continued...

Name	Direction	Width
awvalid	output	1
bid	input	1 - 18
bready	output	1
bresp	input	2
buser	input	1 - 64
bvalid	input	1
rdata	input	8, 16, 32, 64, 128, 256, 512, 1024
rid	input	1 - 18
rlast	input	1
rready	output	1
rresp	input	2
ruser	input	1 - 64
rvalid	input	1
wdata	output	8, 16, 32, 64, 128, 256, 512, 1024
wid	output	1 - 18
wlast	output	1
wready	input	1
wstrb	output	1, 2, 4, 8, 16, 32, 64, 128
wuser	output	1 - 64
wvalid	output	1

11.14.4 AXI4 Slave Interface Signal Types

Table 164. AXI4 Slave Interface Signal Types

Name	Direction	Width
araddr	input	1 - 64
arburst	input	2
arcache	input	4
arid	input	1 - 18
arlen	input	8
arlock	input	1
arprot	input	3
arqos	input	1 - 4
arready	output	1
arregion	input	1 - 4

continued...

Name	Direction	Width
arsize	input	3
aruser	input	1 - 64
arvalid	input	1
awaddr	input	1 - 64
awburst	input	2
awcache	input	4
awid	input	1 - 18
awlen	input	8
awlock	input	1
awprot	input	3
awqos	input	1 - 4
awready	output	1
awregion	inout	1 - 4
awsize	input	3
awuser	input	1 - 64
awvalid	input	1
bid	output	1 - 18
bready	input	1
bresp	output	2
bvalid	output	1
rdata	output	8, 16, 32, 64, 128, 256, 512, 1024
rid	output	1 - 18
rlast	output	1
rready	input	1
rresp	output	2
ruser	output	1 - 64
rvalid	output	1
wdata	input	8, 16, 32, 64, 128, 256, 512, 1024
wlast	input	1
wready	output	1
wstrb	input	1, 2, 4, 8, 16, 32, 64, 128
wuser	input	1 - 64
wvalid	input	1

11.14.5 AXI4 Stream Master and Slave Interface Signal Types

Table 165. AXI4 Stream Master and Slave Interface Signal Types

Name	Width	Master Direction	Slave Direction	Required
tvalid	1	Output	Input	Yes
tready	1	Input	Output	No
tdata	8:4096	Output	Input	No
tstrb	1:512	Output	Input	No
tkeep	1:512	Output	Input	No
tid	1:8	Output	Input	No
tdest	1:4	Output	Input	No
tuser	1	Output	Input	No
tlast	1:4096	Output	Input	No

11.14.6 APB Interface Signal Types

Table 166. APB Interface Signal Types

Name	Width	Direction APB Master	Direction APB Slave	Required
paddr	[1:32]	output	input	yes
psel	[1:16]	output	input	yes
penable	1	output	input	yes
pwrite	1	output	input	yes
pwdata	[1:32]	output	input	yes
prdata	[1:32]	input	output	yes
pslverr	1	input	output	no
pready	1	input	output	yes
paddr31	1	output	input	no

11.14.7 Avalon Memory-Mapped Interface Signal Roles

Signal roles define the signal types that are allowed on Avalon-MM master and slave ports.

This specification does not require all signals to exist in an Avalon-MM interface. There is no one signal that is always required. The minimum requirements for an Avalon-MM interface are `readdata` for a read-only interface, or `writedata` and `write` for a write-only interface.

The following table lists signal roles for the Avalon-MM interface:

Table 167. Avalon-MM Signal Roles

Some Avalon-MM signals can be active high or active low. When active low, the signal name ends with _n.

Signal Role	Width	Direction	Description
Fundamental Signals			
address	1 - 64	Master ☐ Slave	Masters: By default, the address signal represents a byte address. The value of the address must be aligned to the data width. To write to specific bytes within a data word, the master must use the byteenable signal. Refer to the addressUnits interface property for word addressing. Slaves: By default, the interconnect translates the byte address into a word address in the slave's address space. Each slave access is for a word of data from the perspective of the slave. For example, address = 0 selects the first word of the slave. address = 1 selects the second word of the slave. Refer to the addressUnits interface property for byte addressing.
byteenable byteenable_n	2, 4, 8, 16, 32, 64, 128	Master ☐ Slave	Enables specific byte lane(s) during transfers on interfaces of width greater than 8 bits. Each bit in byteenable corresponds to a byte in writedata and readdata. The master bit <n> of byteenable indicates whether byte <n> is being written to. During writes, byteenables specify which bytes are being written to. Other bytes should be ignored by the slave. During reads, byteenables indicate which bytes the master is reading. Slaves that simply return readdata with no side effects are free to ignore byteenables during reads. If an interface does not have a byteenable signal, the transfer proceeds as if all byteenables are asserted. When more than one bit of the byteenable signal is asserted, all asserted lanes are adjacent. The number of adjacent lines must be a power of 2. The specified bytes must be aligned on an address boundary for the size of the data. For example, the following values are legal for a 32-bit slave: • 1111 writes full 32 bits • 0011 writes lower 2 bytes • 1100 writes upper 2 bytes • 0001 writes byte 0 only • 0010 writes byte 1 only • 0100 writes byte 2 only • 1000 writes byte 3 only To avoid unintended side effects, Intel strongly recommends that you use the byteenable signal in systems with different word sizes. *Note:* The AXI interface supports unaligned accesses while Avalon-MM does not. Unaligned accesses going from an AXI master to an Avalon-MM slave may result in an illegal transaction. To avoid this issue, only use aligned accesses to Avalon-MM slaves.
debugaccess	1	Master ☐ Slave	When asserted, allows the Nios II processor to write on-chip memories configured as ROMs.
read read_n	1	Master ☐ Slave	Asserted to indicate a read transfer. If present, readdata is required.

continued...

Signal Role	Width	Direction	Description
`readdata`	8, 16, 32, 64, 128, 256, 512, 1024	Slave ☐ Master	The `readdata` driven from the slave to the master in response to a `read` transfer.
`response [1:0]`	2	Slave ☐ Master	The `response` signal is an optional signal that carries the response status. *Note:* Because the signal is shared, an interface cannot issue or accept a write response and a read response in the same clock cycle. • `00`: OKAY—Successful response for a transaction. • `01`: RESERVED—Encoding is reserved. • `10`: SLAVEERROR—Error from an endpoint slave. Indicates an unsuccessful transaction. • `11`: DECODEERROR—Indicates attempted access to an undefined location. For read responses: • One response is sent with each `readdata`. A read burst length of N results in N responses. It is not valid to produce fewer responses, even in the event of an error. It is valid for the response signal value to be different for each `readdata` in the burst. • The interface must have read control signals. Pipeline support is possible with the `readdatavalid` signal. • On read errors, the corresponding `readdata` is "don't care". For write responses: • One write response must be sent for each write command. A write burst results in only one response, which must be sent after the final write transfer in the burst is accepted. • If `writeresponsevalid` is present, all write commands must be completed with write responses."
`write` `write_n`	1	Master ☐ Slave	Asserted to indicate a `write` transfer. If present, `writedata` is required.
`writedata`	8, 16, 32, 64, 128, 256, 512, 1024	Master ☐ Slave	Data for write transfers. The width must be the same as the width of `readdata` if both are present.
Wait-State Signals			
`lock`	1	Master ☐ Slave	`lock` ensures that once a master wins arbitration, it maintains access to the slave for multiple transactions. It is asserted coincident with the first `read` or `write` of a locked sequence of transactions. It is deasserted on the final transaction of a locked sequence of transactions. `lock` assertion does not guarantee that arbitration will be won. After the lock-asserting master has been granted, it retains grant until it is deasserted. A master equipped with `lock` cannot be a burst master. Arbitration priority values for lock-equipped masters are ignored. `lock` is particularly useful for read-modify-write (RMW) operations. The typical read-modify-write operation includes the following steps: 1. Master A asserts lock and reads 32-bit data that has multiple bit fields. 2. Master A deasserts lock, changes one bit field, and writes the 32-bit data back.

continued...

Signal Role	Width	Direction	Description
			`lock` prevents master B from performing a write between Master A's read and write.
`waitrequest` `waitrequest_n`	1	Slave ☐ Master	Asserted by the slave when it is unable to respond to a `read` or `write` request. Forces the master to wait until the interconnect is ready to proceed with the transfer. At the start of all transfers, a master initiates the transfer and waits until `waitrequest` is deasserted. A master must make no assumption about the assertion state of `waitrequest` when the master is idle: `waitrequest` may be high or low, depending on system properties. When `waitrequest` is asserted, master control signals to the slave must remain constant with the exception of `beginbursttransfer`. For a timing diagram illustrating the `beginbursttransfer` signal, refer to the figure in *Read Bursts*. An Avalon-MM slave may assert `waitrequest` during idle cycles. An Avalon-MM master may initiate a transaction when `waitrequest` is asserted and wait for that signal to be deasserted. To avoid system lockup, a slave device should assert `waitrequest` when in reset.
Pipeline Signals			
`readdatavalid` `readdatavalid_n`	1	Slave ☐ Master	Used for variable-latency, pipelined `read` transfers. When asserted, indicates that the `readdata` signal contains valid data. For a read burst with burstcount value <*n*>, the `readdatavalid` signal must be asserted <*n*> times, once for each readdata item. There must be at least one cycle of latency between acceptance of the `read` and assertion of `readdatavalid`. For a timing diagram illustrating the `readdatavalid` signal, refer to *Pipelined Read Transfer with Variable Latency*. A slave may assert `readdatavalid` to transfer data to the master independently of whether or not the slave is stalling a new command with `waitrequest`. Required if the master supports pipelined reads. Bursting masters with read functionality must include the `readdatavalid` signal.
`writeresponsevalid`			An optional signal. If present, the interface issues write responses for write commands. When asserted, the value on the response signal is a valid write response. `Writeresponsevalid` is only asserted one clock cycle or more after the write command is accepted. There is at least a one clock cycle latency from command acceptance to assertion of `writeresponsevalid`.
Burst Signals			
`burstcount`	1 – 11	Master ☐ Slave	Used by bursting masters to indicate the number of transfers in each burst. The value of the maximum `burstcount` parameter must be a power of 2. A burstcount interface of width <n> can encode a max burst of size $2^{(<n>-1)}$. For example, a 4-bit `burstcount` signal can support a maximum burst count of 8. The minimum `burstcount` is 1. The `constantBurstBehavior` property controls the

continued...

Signal Role	Width	Direction	Description
			timing of the `burstcount` signal. Bursting masters with read functionality must include the `readdatavalid` signal. For bursting masters and slaves using byte addresses, the following restriction applies to the width of the address: *<address_w> >= <burstcount_w>* $+\log_2$*(<symbols_per_word_of_interface>)*. For bursting masters and slaves using word addresses, the $\log_2$ term above is omitted.
`beginbursttransfer`	1	Interconnect ☐ Slave	Asserted for the first cycle of a burst to indicate when a burst transfer is starting. This signal is deasserted after one cycle regardless of the value of `waitrequest`. For a timing diagram illustrating `beginbursttransfer`, refer to the figure in *Read Bursts*. `beginbursttransfer` is optional. A slave can always internally calculate the start of the next write burst transaction by counting data transfers. Intel recommends that you *do not* use this signal. This signal exists to support legacy memory controllers.

11.14.8 Avalon Streaming Interface Signal Roles

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

Table 168. Avalon-ST Interface Signals

In the following table, all signal roles are active high.

Signal Role	Width	Direction	Description
Fundamental Signals			
`channel`	1 – 128	Source ☐ Sink	The `channel` number for data being transferred on the current cycle. If an interface supports the channel signal, it must also define the `maxChannel` parameter.
`data`	1 – 4,096	Source ☐ Sink	The `data` signal from the source to the sink, typically carries the bulk of the information being transferred. The contents and format of the `data` signal is further defined by parameters.
`error`	1 – 256	Source ☐ Sink	A bit mask used to mark errors affecting the data being transferred in the current cycle. A single bit in `error` is used for each of the errors recognized by the component, as defined by the `errorDescriptor` property.
`ready`	1	Sink ☐ Source	Asserted high to indicate that the sink can accept data. `ready` is asserted by the sink on cycle <n> to mark cycle <*n* + `readyLatency`> as a ready cycle. The source may only assert `valid` and transfer data during `ready` cycles. Sources without a `ready` input cannot be backpressured. Sinks without a `ready` output never need to backpressure.

continued...

Signal Role	Width	Direction	Description
valid	1	Source ☐ Sink	Asserted by the source to qualify all other source to sink signals. The sink samples data and other source-to-sink signals on ready cycles where valid is asserted. All other cycles are ignored. Sources without a valid output implicitly provide valid data on every cycle that they are not being backpressured. Sinks without a valid input expect valid data on every cycle that they are not backpressuring.
Packet Transfer Signals			
empty	1 – 5	Source ☐ Sink	Indicates the number of symbols that are empty, that is, do not represent valid data. The empty signal is not used on interfaces where there is one symbol per beat.
endofpacket	1	Source ☐ Sink	Asserted by the source to mark the end of a packet.
startofpacket	1	Source ☐ Sink	Asserted by the source to mark the beginning of a packet.

11.14.9 Avalon Clock Source Signal Roles

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

Table 169. Clock Source Signal Roles

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

11.14.10 Avalon Clock Sink Signal Roles

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

Table 170. Clock Sink Signal Roles

Signal Role	Width	Direction	Required	Description
clk	1	Input	Yes	A clock signal. Provides synchronization for internal logic and for other interfaces.

11.14.11 Avalon Conduit Signal Roles

Table 171. Conduit Signal Roles

Signal Role	Width	Direction	Description
<any>	<*n*>	In, out, or bidirectional	A conduit interface consists of one or more input, output, or bidirectional signals of arbitrary width. Conduits can have any user-specified role. You can connect compatible Conduit interfaces inside a Qsys Pro system provided the roles and widths match and the directions are opposite.

11.14.12 Avalon Tristate Conduit Signal Roles

The following table lists the signal defined for the Avalon Tristate Conduit interface. All Avalon-TC signals apply to both masters and slaves and have the same meaning for both

Table 172. Tristate Conduit Interface Signal Roles

Signal Role	Width	Direction	Required	Description
`request`	1	Master ☐ Slave	Yes	The meaning of `request` depends on the state of the `grant` signal, as the following rules dictate. When `request` is asserted and `grant` is deasserted, `request` is requesting access for the current cycle. When `request` is asserted and `grant` is asserted, `request` is requesting access for the next cycle. Consequently, `request` should be deasserted on the final cycle of an access. The `request` is deasserted in the last cycle of a bus access. It can be reasserted immediately following the final cycle of a transfer. This protocol makes both rearbitration and continuous bus access possible if no other masters are requesting access. Once asserted, `request` must remain asserted until granted. Consequently, the shortest bus access is 2 cycles. Refer to *Tristate Conduit Arbitration Timing* for an example of arbitration timing.
`grant`	1	Slave ☐ Master	Yes	When asserted, indicates that a tristate conduit master has been granted access to perform transactions. `grant` is asserted in response to the `request` signal. It remains asserted until 1 cycle following the deassertion of `request`.
<*name*>_in	1 – 1024	Slave ☐ Master	No	The input signal of a logical tristate signal.
<*name*>_out	1 – 1024	Master ☐ Slave	No	The output signal of a logical tristate signal.
<*name*>_outen	1	Master ☐ Slave	No	The output enable for a logical tristate signal.

11.14.13 Avalon Tri-State Slave Interface Signal Types

Table 173. Tri-state Slave Interface Signal Types

Name	Width	Direction	Required	Description
address	1 - 32	input	No	Address lines to the slave port. Specifies a byte offset into the slave's address space.
read read_n	1	input	No	Read-request signal. Not required if the slave port never outputs data. If present, data must also be used.
write write_n	1	input	No	Write-request signal. Not required if the slave port never receives data from a master. If present, data must also be present, and writebyteenable cannot be present.
chipselect chipselect_n	1	input	No	When present, the slave port ignores all Avalon-MM signals unless chipselect is asserted. chipselect is always present in combination with read or write
outputenable outputenable_n	1	input	Yes	Output-enable signal. When deasserted, a tri-state slave port must not drive its data lines otherwise data contention may occur.
data	8,16, 32, 64, 128, 256, 512, 1024	bidir	No	Bidirectional data. During write transfers, the FPGA drives the data lines. During read transfers the slave device drives the data lines, and the FPGA captures the data signals and provides them to the master.

byteenable
byteenable_n
	2, 4, 8,16, 32, 64, 128	input	No	Enables specific byte lane(s) during transfers.
				Each bit in byteenable corresponds to a byte lane in data. During writes, byteenables specify which bytes the master is writing to the slave. During reads, byteenables indicates which bytes the master is reading. Slaves that simply return data with no side effects are free to ignore byteenables during reads.
				When more than one byte lane is asserted, all asserted lanes are guaranteed to be adjacent. The number of adjacent lines must be a power of 2, and the specified bytes must be aligned on an address boundary for the size of the data. The are legal values for a 32-bit slave:
				`1111 writes full 32 bits`
				`0011 writes lower 2 bytes`
				`1100 writes upper 2 bytes`
				`0001 writes byte 0 only`
				`0010 writes byte 1 only`
				`0100 writes byte 2 only`
				`1000 writes byte 3 only`

continued...

Name	Width	Direction	Required	Description
writebyteenable writebyteenable_n	2,4,8,16, 32, 64,128	input	No	Equivalent to the logical AND of the byteenable and write signals. When used, the write signal is not used.
begintransfer1	1	input	No	Asserted for the first cycle of each transfer.
Note: All Avalon signals are active high. Avalon signals that can also be asserted low list both versions in the **Signal Role** column.				

11.14.14 Avalon Interrupt Sender Signal Roles

Table 174. Interrupt Sender Signal Roles

Signal Role	Width	Direction	Required	Description
irq irq_n	1	Output	Yes	Interrupt Request. A slave asserts irq when it needs service. The interrupt receiver determines the relative priority of the interrupts.

11.14.15 Avalon Interrupt Receiver Signal Roles

Table 175. Interrupt Receiver Signal Roles

Signal Role	Width	Direction	Required	Description
irq	1–32	Input	Yes	irq is an <*n*>-bit vector, where each bit corresponds directly to one IRQ sender with no inherent assumption of priority.

11.15 Document Revision History

The table below indicates edits made to the *Qsys Pro Interconnect* content since its creation.

Table 176. Document Revision History

Date	Version	Changes
2016.10.31	16.1.0	• Implemented Intel rebranding. • Implemented Qsys Pro rebranding.
2015.11.02	15.1.0	Changed instances of *Quartus II* to *Quartus Prime*.
2015.05.04	15.0.0	• Fixed Priority Arbitration. • Added topic: *Read and Write Responses*. • Added topic: *Error Correction Coding (ECC) in Qsys Pro Interconnect*. • Added: response [1:0], *Avalon Memory-Mapped Interface Signal Roles*. • Added writeresponsevalid, *Avalon Memory-Mapped Interface Signal Roles*.
December 2014	14.1.0	• Read error responses, Avalon Memory-Mapped Interface Signal, response. • Burst Adapter Implementation Options: Generic converter (slower, lower area), Per-burst-type converter (faster, higher area).

continued...

Date	Version	Changes
August 2014	14.0a10.0	• Updated Qsys Pro Packet Format for Memory-Mapped Master and Slave Interfaces table, `Protection`. • Streaming Interface renamed to Avalon Streaming Interfaces. • Added *Response Merging* under *Memory-Mapped Interfaces*.
June 2014	14.0.0	• AXI4-Lite support. • AXI4-Stream support. • Avalon-ST adapter parameters. • IRQ Bridge. • Handling Read Side Effects note added.
November 2013	13.1.0	• HSSI clock support. • Reset Sequencer. • Interconnect pipelining.
May 2013	13.0.0	• AMBA APB support. • Auto-inserted Avalon-ST adapters feature. • Moved Address Span Extender to the *Qsys Pro System Design Components* chapter.
November 2012	12.1.0	• AMBA AXI4 support.
June 2012	12.0.0	• AMBA AXI3 support. • Avalon-ST adapters. • Address Span Extender.
November 2011	11.0.1	Template update.
May 2011	11.0.0	Removed beta status.
December 2010	10.1.0	Initial release.

Related Links

Altera Documentation Archive

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

12 Optimizing Qsys Pro System Performance

You can optimize system interconnect performance for Intel designs that you create with the Qsys Pro system integration tool.

The foundation of any system is the interconnect logic that connects hardware blocks or components. Creating interconnect logic is prone to errors, is time consuming to write, and is difficult to modify when design requirements change. The Qsys Pro system integration tool addresses these issues and provides an automatically generated and optimized interconnect designed to satisfy your system requirements.

Qsys Pro supports Avalon, AMBA AXI3 (version 1.0), AMBA AXI4 (version 2.0), AMBA AXI4-Lite (version 2.0), AMBA AXI4-Stream (version 1.0), and AMBA APB3 (version 1.0) interface specifications.

Note: Recommended Intel practices may improve clock frequency, throughput, logic utilization, or power consumption of your Qsys Pro design. When you design a Qsys Pro system, use your knowledge of your design intent and goals to further optimize system performance beyond the automated optimization available in Qsys Pro.

Related Links

- Creating a System With Qsys Pro on page 299
  Qsys Pro is a system integration tool included as part of the Quartus Prime software.
- Creating Qsys Pro Components on page 585
  You can create a Hardware Component Definition File (`_hw.tcl`) to describe and package IP components for use in a Qsys Pro system.
- Qsys Pro Interconnect on page 627
  Qsys Pro interconnect is a high-bandwidth structure that allows you to connect IP components to other IP components with various interfaces.
- Avalon Interface Specifications
- AMBA Protocol Specifications

12.1 Designing with Avalon and AXI Interfaces

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

Avalon Streaming (Avalon-ST) links connect point-to-point, unidirectional interfaces and are typically used in data stream applications. Each pair of components is connected without any requirement to arbitrate between the data source and sink.

Because Qsys Pro supports multiplexed memory-mapped and streaming connections, you can implement systems that use multiplexed logic for control and streaming for data in a single design.

Related Links

Creating Qsys Pro Components on page 585
You can create a Hardware Component Definition File (_hw.tcl) to describe and package IP components for use in a Qsys Pro system.

12.1.1 Designing Streaming Components

When you design streaming component interfaces, you must consider integration and communication for each component in the system. One common consideration is buffering data internally to accommodate latency between components.

For example, if the component's Avalon-ST output or source of streaming data is back-pressured because the ready signal is deasserted, then the component must back-pressure its input or sink interface to avoid overflow.

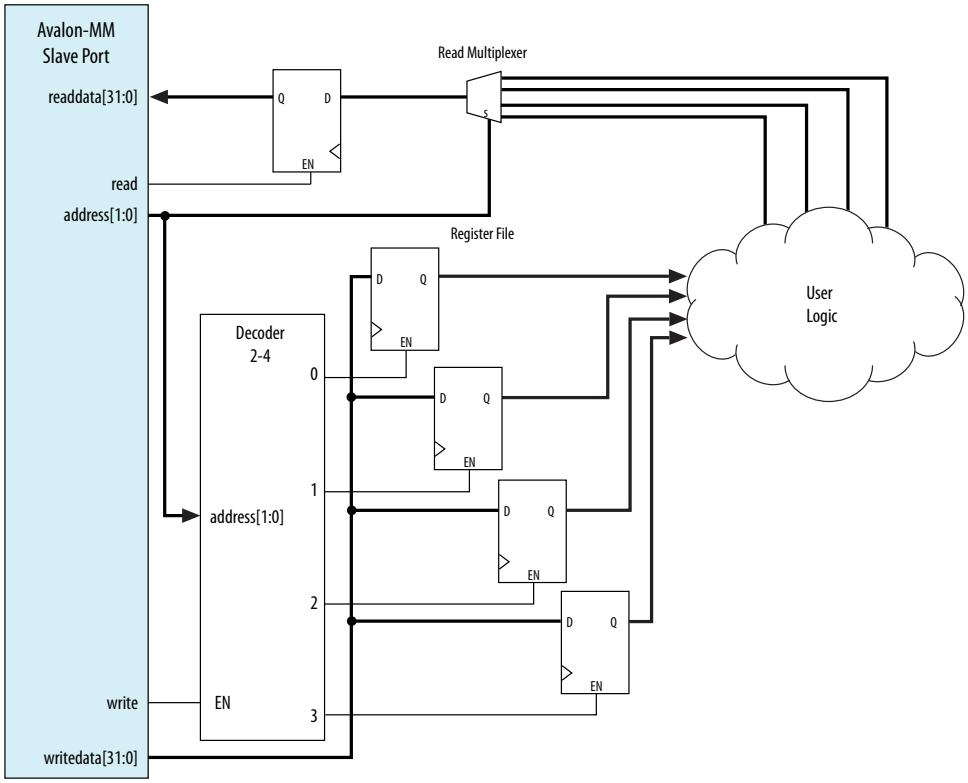
You can use a FIFO to back-pressure internally on the output side of the component so that the input can accept more data even if the output is back-pressured. Then, you can use the FIFO almost full flag to back-pressure the sink interface or input data when the FIFO has only enough space to satisfy the internal latency. You can drive the data valid signal of the output or source interface with the FIFO not empty flag when that data is available.

12.1.2 Designing Memory-Mapped Components

When designing with memory-mapped components, you can implement any component that contains multiple registers mapped to memory locations, for example, a set of four output registers to support software read back from logic. Components that implement read and write memory-mapped transactions require three main building blocks: an address decoder, a register file, and a read multiplexer.

The decoder enables the appropriate 32-bit or 64-bit register for writes. For reads, the address bits drive the multiplexer selection bits. The read signal registers the data from the multiplexer, adding a pipeline stage so that the component can achieve a higher clock frequency.

Figure 209. Control and Status Registers (CSR) in a Slave Component



This slave component has four write wait states and one read wait state. Alternatively, if you want high throughput, you may set both the read and write wait states to zero, and then specify a read latency of one, because the component also supports pipelined reads.

12.2 Using Hierarchy in Systems

You can use hierarchy to sub-divide a system into smaller subsystems that you can then connect in a top-level Qsys Pro system. Additionally, if a design contains one or more identical functional units, the functional unit can be defined as a subsystem and instantiated multiple times within a top-level system.

Hierarchy can simplify verification control of slaves connected to each master in a memory-mapped system. Before you implement subsystems in your design, you should plan the system hierarchical blocks at the top-level, using the following guidelines:

- **Plan shared resources**—Determine the best location for shared resources in the system hierarchy. For example, if two subsystems share resources, add the components that use those resources to a higher-level system for easy access.
- **Plan shared address space between subsystems**—Planning the address space ensures you can set appropriate sizes for bridges between subsystems.
- **Plan how much latency you may need to add to your system**—When you add an Avalon-MM Pipeline Bridge between subsystems, you may add latency to the overall system. You can reduce the added latency by parameterizing the bridge with zero cycles of latency, and by turning off the pipeline command and response signals.

Figure 210. Avalon-MM Pipeline Bridge

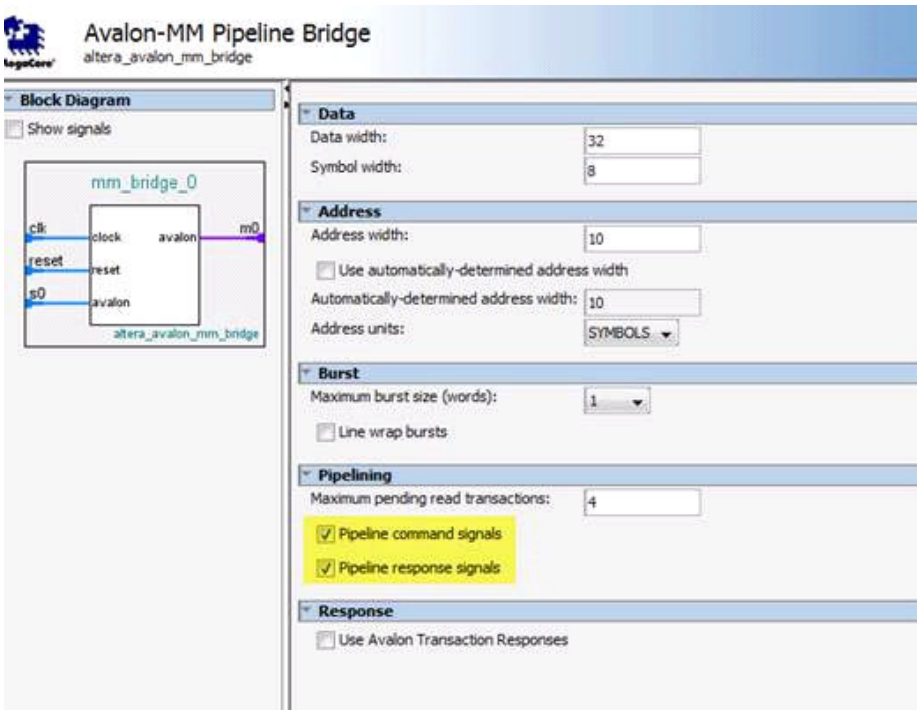
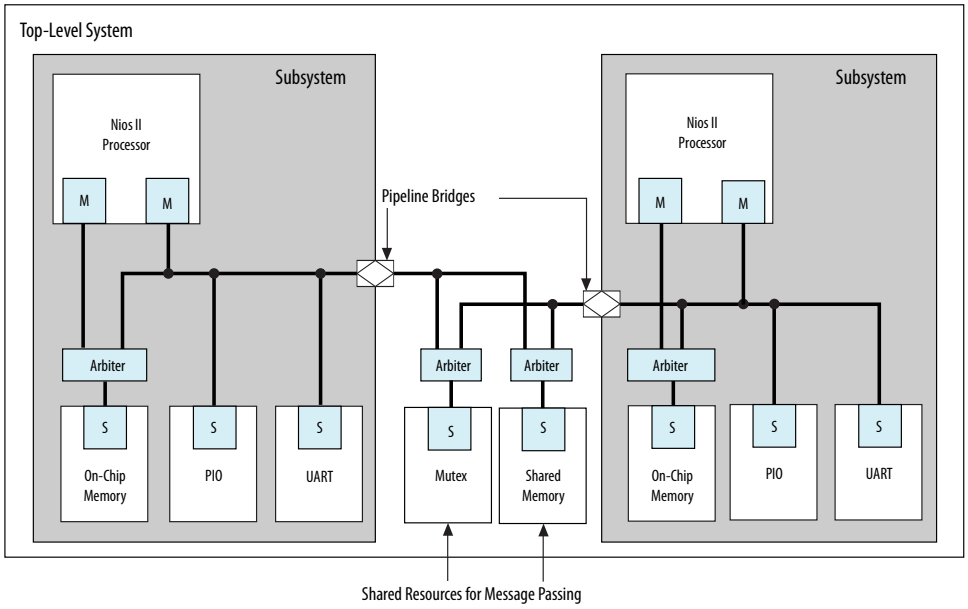
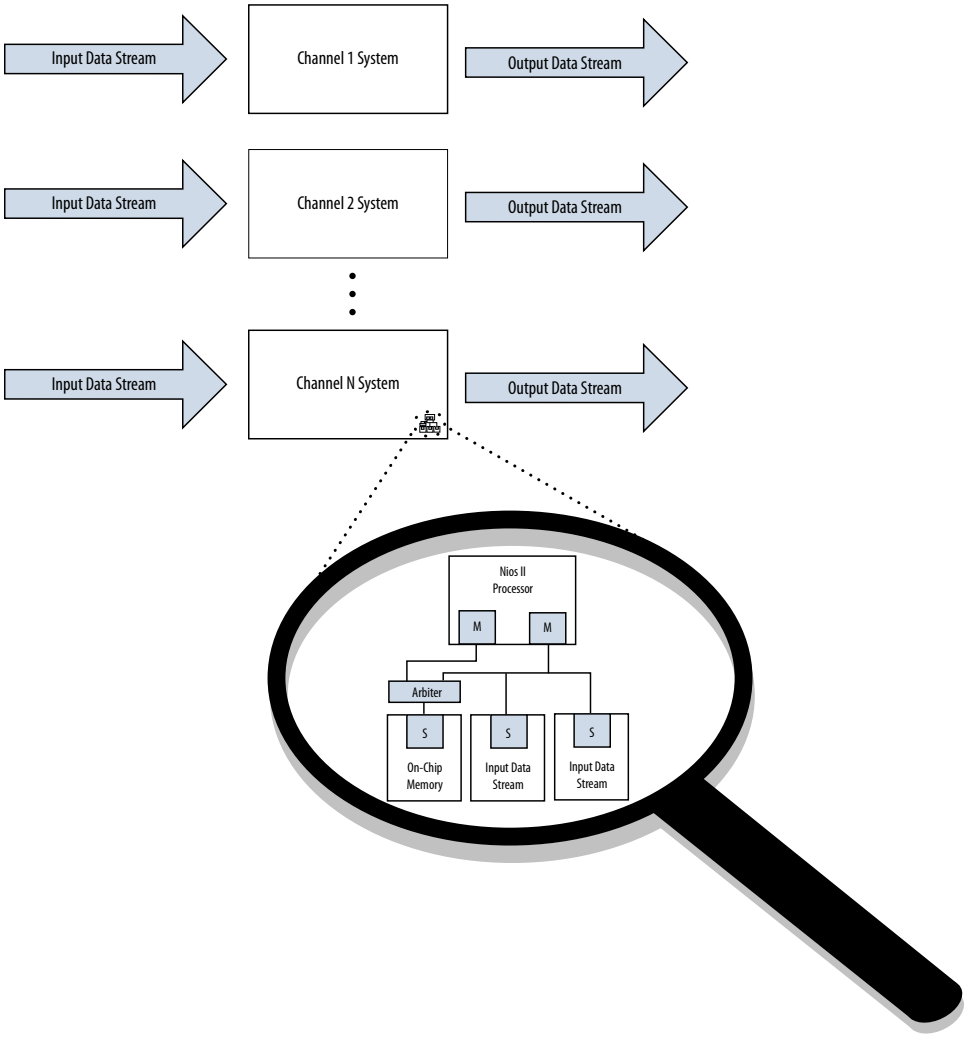


Figure 211. Passing Messages Between Subsystems



In this example, two Nios II processor subsystems share resources for message passing. Bridges in each subsystem export the Nios II data master to the top-level system that includes the mutex (mutual exclusion component) and shared memory component (which could be another on-chip RAM, or a controller for an off-chip RAM device).

Figure 212. Multi Channel System



You can also design systems that process multiple data channels by instantiating the same subsystem for each channel. This approach is easier to maintain than a larger, non-hierarchical system. Additionally, such systems are easier to scale because you can calculate the required resources as a multiple of the subsystem requirements.

Related Links

Avalon-MM Pipeline Bridge

12.3 Using Concurrency in Memory-Mapped Systems

Qsys Pro interconnect uses parallel hardware in FPGAs, which allows you to design concurrency into your system and process transactions simultaneously.

12.3.1 Implementing Concurrency With Multiple Masters

Implementing concurrency requires multiple masters in a Qsys Pro system. Systems that include a processor contain at least two master interfaces because the processors include separate instruction and data masters. You can catagorize master components as follows:

- General purpose processors, such as Nios II processors
- DMA (direct memory access) engines
- Communication interfaces, such as PCI Express

Because Qsys Pro generates an interconnect with slave-side arbitration, every master interface in a system can issue transfers concurrently, as long as they are not posting transfers to the same slave. Concurrency is limited by the number of master interfaces sharing any particular slave interface. If a design requires higher data throughput, you can increase the number of master and slave interfaces to increase the number of transfers that occur simultaneously. The example below shows a system with three master interfaces.

Figure 213. Avalon Multiple Master Parallel Access

In this Avalon example, the DMA engine operates with Avalon-MM read and write masters. However, an AXI DMA interface typically has only one master, because in the AXI standard, the write and read channels on the master are independent and can process transactions simultaneously. The yellow lines represent active simultaneously connections.

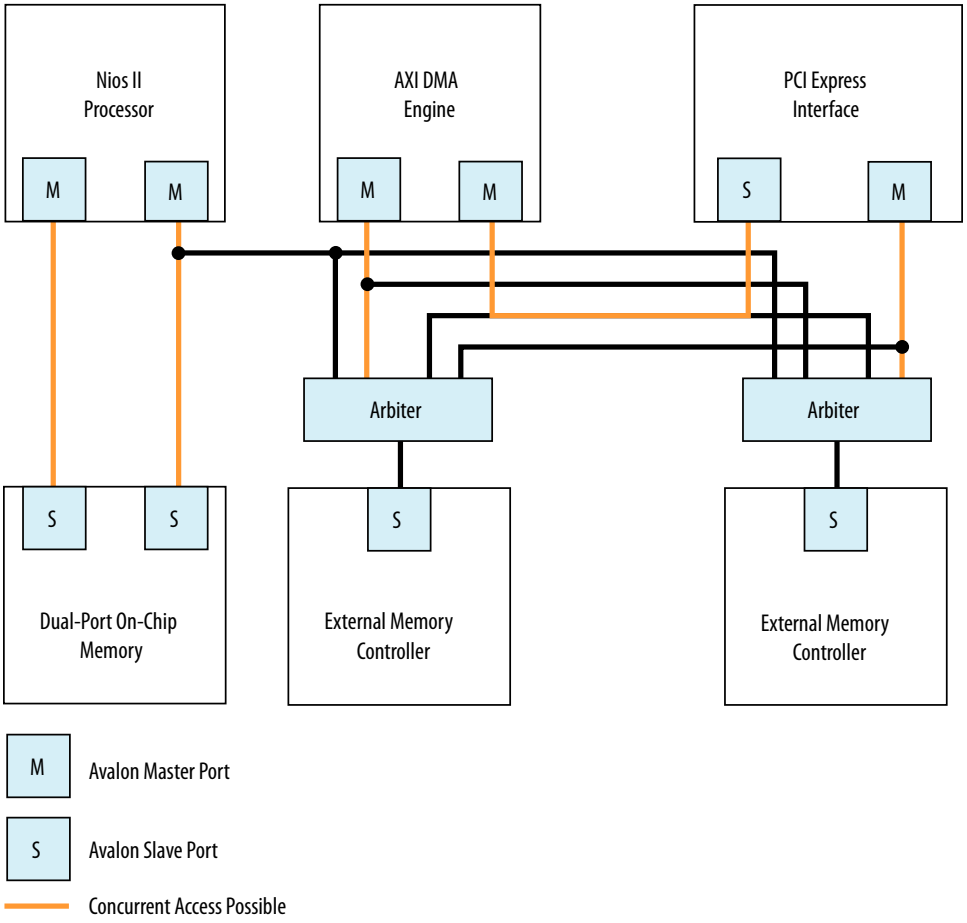
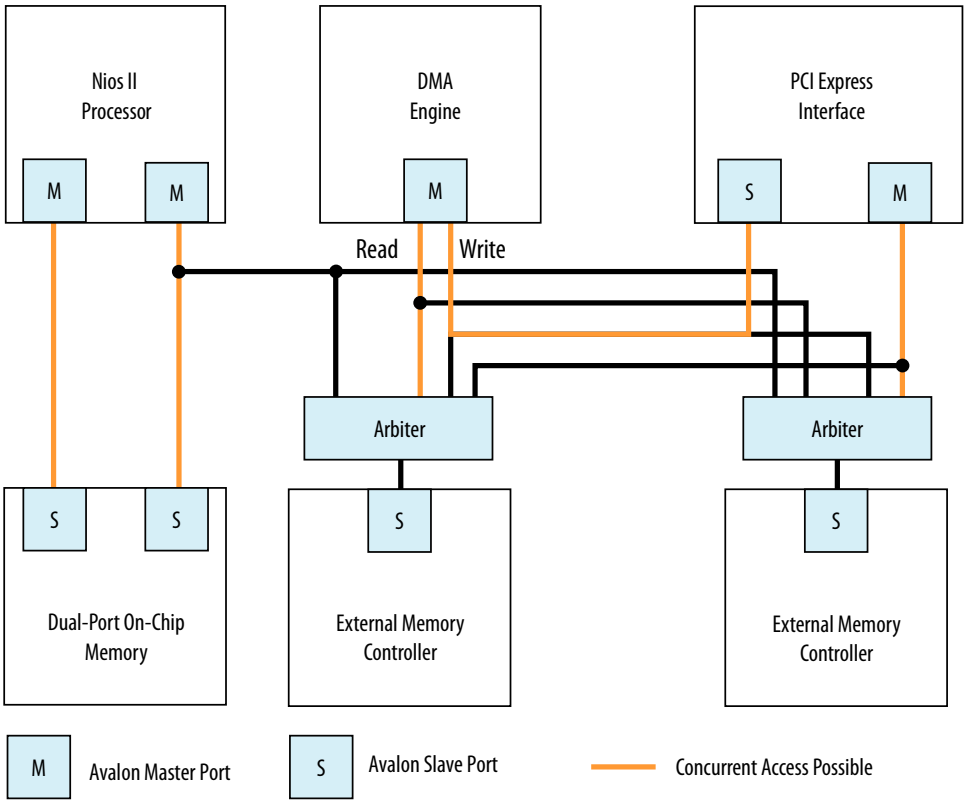


Figure 214. AXI Multiple Master Parallel Access

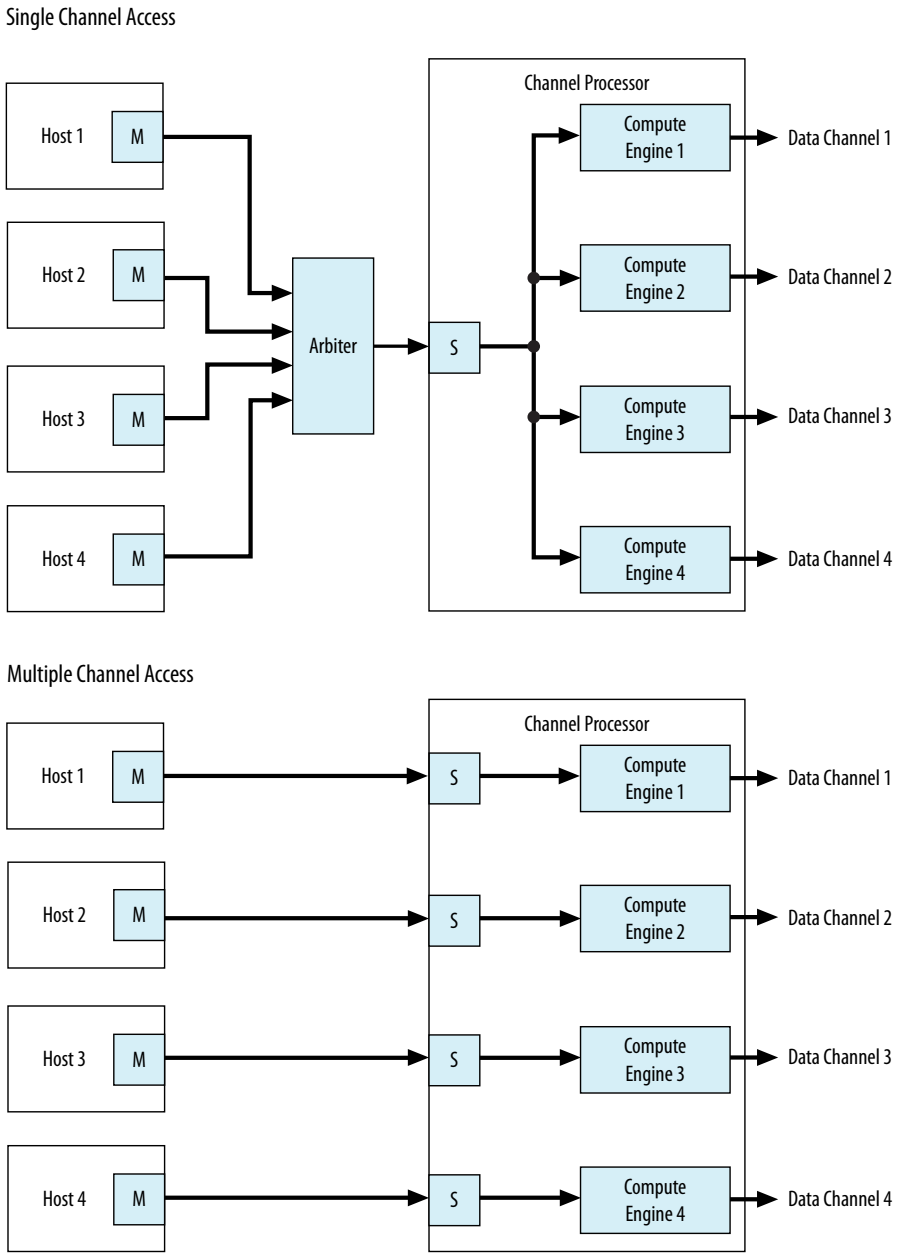
In this example, the DMA engine operates with a single master, because in AXI, the write and read channels on the master are independent and can process transactions simultaneously. There is concurrency between the read and write channels, with the yellow lines representing concurrent datapaths.



12.3.2 Implementing Concurrency With Multiple Slaves

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

Figure 215. Single Interface Versus Multiple Interfaces

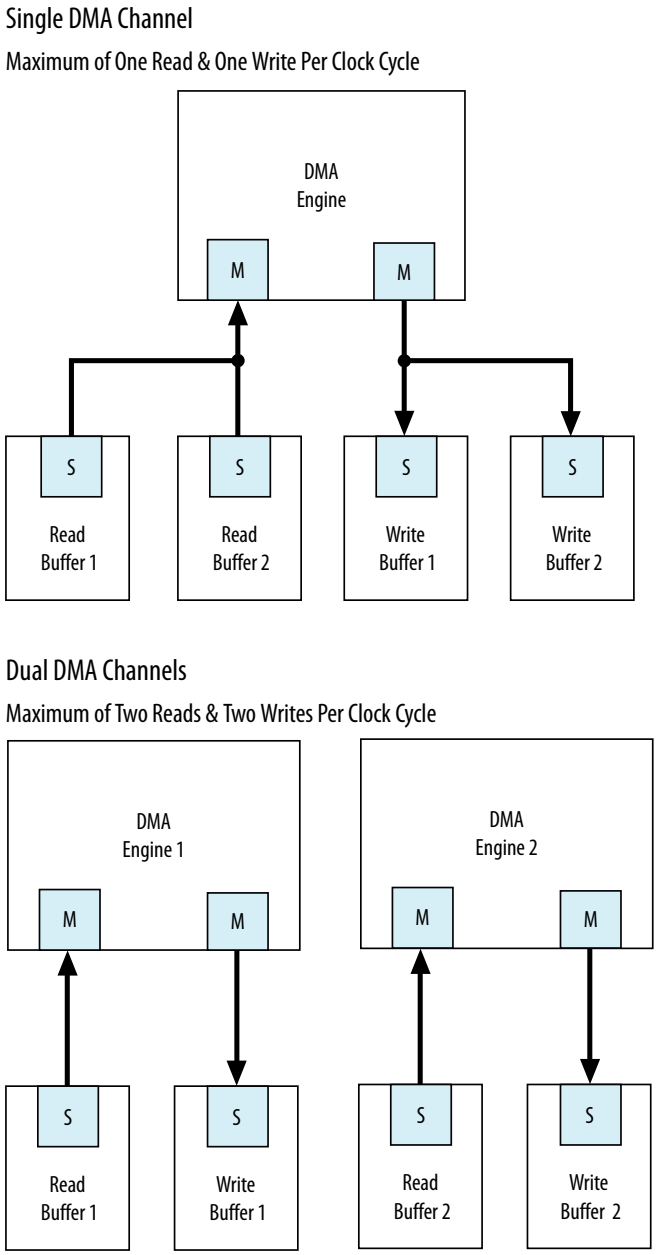


In this example, there are two channel processing systems. In the first, four hosts must arbitrate for the single slave interface of the channel processor. In the second, each host drives a dedicated slave interface, allowing all master interfaces to simultaneously access the slave interfaces of the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

12.3.3 Implementing Concurrency with DMA Engines

In some systems, you can use DMA engines to increase throughput. You can use a DMA engine to transfer blocks of data between interfaces, which then frees the CPU from doing this task. A DMA engine transfers data between a programmed start and end address without intervention, and the data throughput is dictated by the components connected to the DMA. Factors that affect data throughput include data width and clock frequency.

Figure 216. Single or Dual DMA Channels



In this example, the system can sustain more concurrent read and write operations by including more DMA engines. Accesses to the read and write buffers in the top system are split between two DMA engines, as shown in the Dual DMA Channels at the bottom of the figure.

The DMA engine operates with Avalon-MM write and read masters. An AXI DMA typically has only one master, because in AXI, the write and read channels on the master are independent and can process transactions simultaneously.

12.4 Inserting Pipeline Stages to Increase System Frequency

Qsys Pro provides the **Limit interconnect pipeline stages to** option on the **Project Settings** tab to automatically add pipeline stages to the Qsys Pro interconnect when you generate a system.

You can specify between 0 to 4 pipeline stages, where 0 means that the interconnect has a combinational datapath. You can specify a unique interconnect pipeline stage value for each subsystem.

Adding pipeline stages may increase the f_{MAX} of the design by reducing the combinational logic depth, at the cost of additional latency and logic utilization.

The insertion of pipeline stages requires certain interconnect components. For example, in a system with a single slave interface, there is no multiplexer; therefore multiplexer pipelining does not occur. When there is an Avalon or AXI single-master to single-slave system, no pipelining occurs, regardless of the **Limit interconnect pipeline stages to** option.

Related Links

- Interconnect Pipelining on page 684
  Qsys Pro can automatically insert Avalon-ST pipeline stages when you generate your design.
- Pipelined Avalon-MM Interfaces on page 740
- Creating a System with Qsys Pro on page 299
  Qsys Pro is a system integration tool included as part of the Quartus Prime software.

12.5 Using Bridges

You can use bridges to increase system frequency, minimize generated Qsys Pro logic, minimize adapter logic, and to structure system topology when you want to control where Qsys Pro adds pipelining. You can also use bridges with arbiters when there is concurrency in the system.

An Avalon bridge has an Avalon-MM slave interface and an Avalon-MM master interface. You can have many components connected to the bridge slave interface, or many components connected to the bridge master interface. You can also have a single component connected to a single bridge slave or master interface.

You can configure the data width of the bridge, which can affect how Qsys Pro generates bus sizing logic in the interconnect. Both interfaces support Avalon-MM pipelined transfers with variable latency, and can also support configurable burst lengths.

Transfers to the bridge slave interface are propagated to the master interface, which connects to components downstream from the bridge. When you need greater control over interconnect pipelining, you can use bridges instead of the **Limit Interconnect Pipeline Stages to** option.

Note: You can use Avalon bridges between AXI interfaces, and between Avalon domains. Qsys Pro automatically creates interconnect logic between the AXI and Avalon interfaces, so you do not have to explicitly instantiate bridges between these domains. For more discussion about the benefits and disadvantages of shared and separate domains, refer to the *Qsys Pro Interconnect*.

Related Links

- Bridges on page 879
  Bridges affect the way Qsys Pro transports data between components. You can insert bridges between master and slave interfaces to control the topology of a Qsys Pro system, which affects the interconnect that Qsys Pro generates. You can also use bridges to separate components into different clock domains to isolate clock domain crossing logic.
- Bridges on page 691
  With APB, you cannot use bridge components that use multiple PSELx in Qsys Pro. As a workaround, you can group PSELx, and then send the packet to the slave directly.

12.5.1 Using Bridges to Increase System Frequency

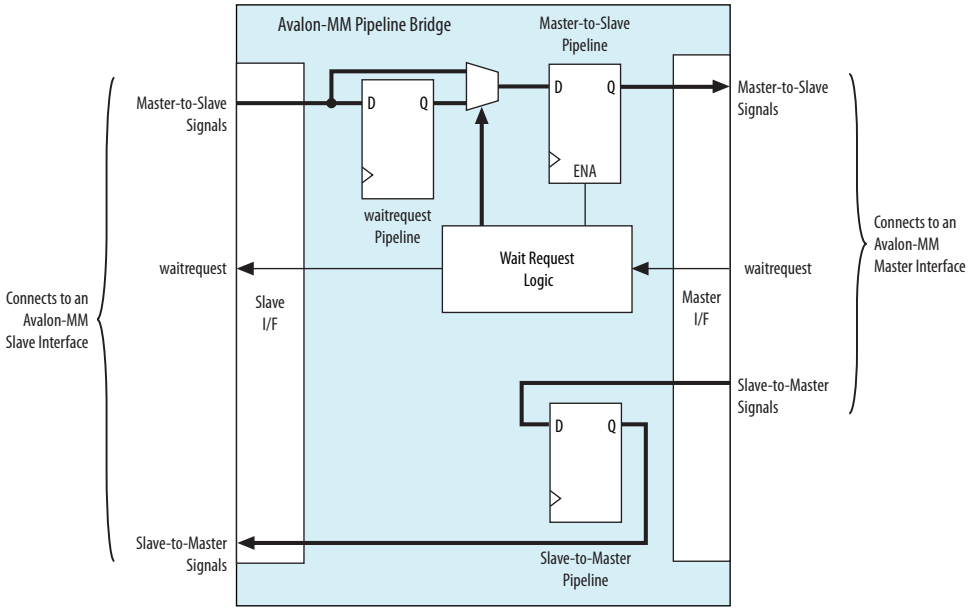
In Qsys Pro, you can introduce interconnect pipeline stages or pipeline bridges to increase clock frequency in your system. Bridges control the system interconnect topology and allow you to subdivide the interconnect, giving you more control over pipelining and clock crossing functionality.

12.5.1.1 Inserting Pipeline Bridges

You can insert an Avalon-MM pipeline bridge to insert registers in the path between the bridges and its master and slaves. If a critical register-to-register delay occurs in the interconnect, a pipeline bridge can help reduce this delay and improve system f_{MAX}.

The Avalon-MM pipeline bridge component integrates into any Qsys Pro system. The pipeline bridge options can increase logic utilization and read latency. The change in topology may also reduce concurrency if multiple masters arbitrate for the bridge. You can use the Avalon-MM pipeline bridge to control topology without adding a pipeline stage. A pipeline bridge that does not add a pipeline stage is optimal in some latency-sensitive applications. For example, a CPU may benefit from minimal latency when accessing memory.

Figure 217. Avalon-MM Pipeline Bridge

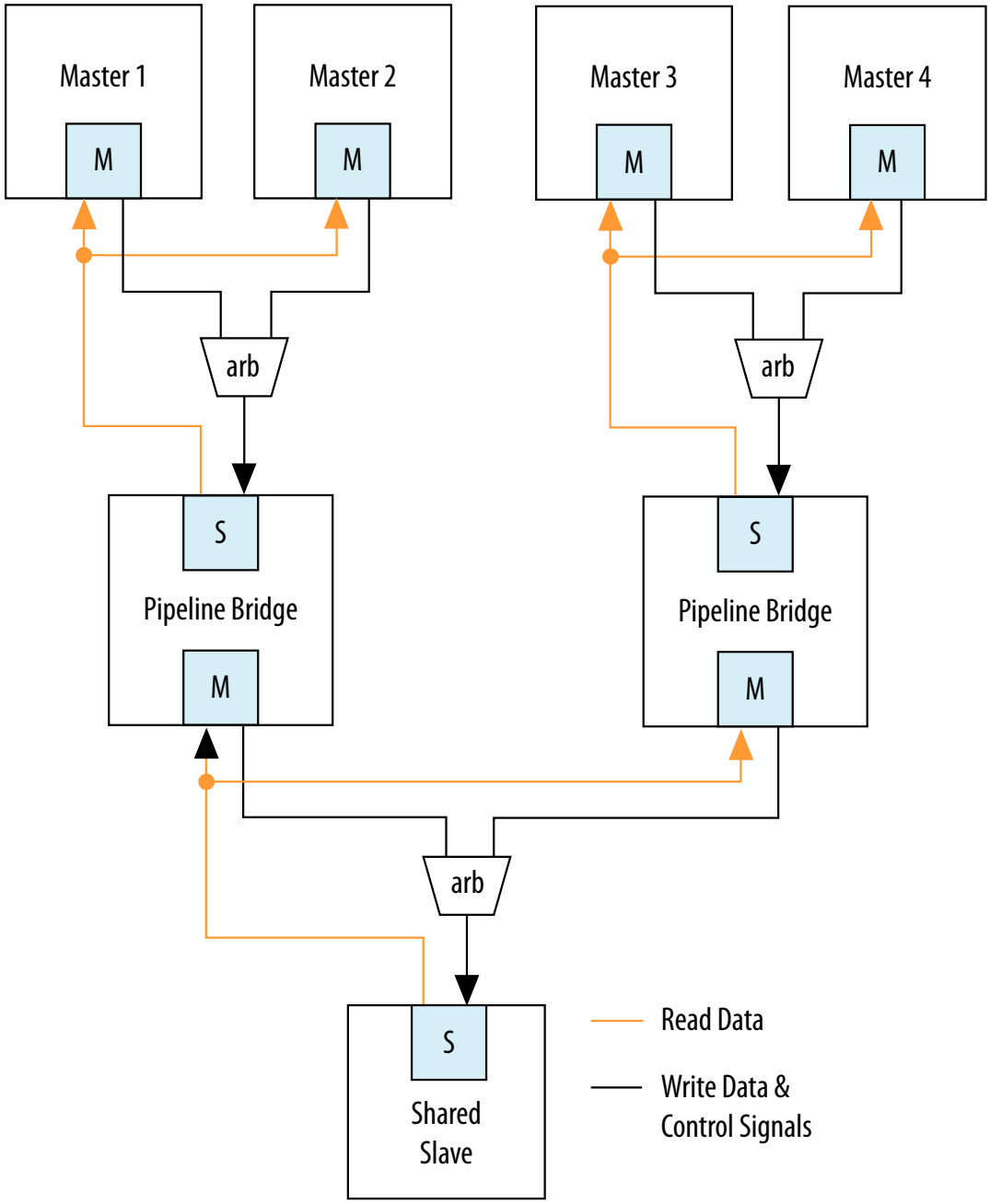


12.5.1.1.1 Implementing Command Pipelining (Master-to-Slave)

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

The arbitration logic for the slave interface must multiplex the `address`, `writedata`, and `burstcount` signals. The multiplexer width increases proportionally with the number of masters connecting to a single slave interface. The increased multiplexer width may become a timing critical path in the system. If a single pipeline bridge does not provide enough pipelining, you can instantiate multiple instances of the bridge in a tree structure to increase the pipelining and further reduce the width of the multiplexer at the slave interface.

Figure 218. Tree of Bridges



12.5.1.1.2 Implementing Response Pipelining (Slave-to-Master)

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

The interconnect inserts a multiplexer for every read datapath back to the master. As the number of slaves supporting read transfers connecting to the master increases, the width of the read data multiplexer also increases. If the performance increase is insufficient with one bridge, you can use multiple bridges in a tree structure to improve f_{MAX}.

12.5.1.2 Using Clock Crossing Bridges

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

When you use a FIFO clock crossing bridge for the clock domain crossing, you add data buffering. Buffering allows pipelined read masters to post multiple reads to the bridge, even if the slaves downstream from the bridge do not support pipelined transfers.

You can also use a clock crossing bridge to place high and low frequency components in separate clock domains. If you limit the fast clock domain to the portion of your design that requires high performance, you may achieve a higher f_{MAX} for this portion of the design. For example, the majority of processor peripherals in embedded designs do not need to operate at high frequencies, therefore, you do not need to use a high-frequency clock for these components. When you compile a design with the Quartus Prime software, compilation may take more time when the clock frequency requirements are difficult to meet because the Fitter needs more time to place registers to achieve the required f_{MAX}. To reduce the amount of effort that the Fitter uses on low priority and low performance components, you can place these behind a clock crossing bridge operating at a lower frequency, allowing the Fitter to increase the effort placed on the higher priority and higher frequency datapaths.

12.5.2 Using Bridges to Minimize Design Logic

Bridges can reduce interconnect logic by reducing the amount of arbitration and multiplexer logic that Qsys Pro generates. This reduction occurs because bridges limit the number of concurrent transfers that can occur.

12.5.2.1 Avoiding Speed Optimizations That Increase Logic

You can add an additional pipeline stage with a pipeline bridge between masters and slaves to reduce the amount of combinational logic between registers, which can increase system performance. If you can increase the f_{MAX} of your design logic, you may be able to turn off the Quartus Prime software optimization settings, such as the **Perform register duplication** setting. Register duplication creates duplicate registers in two or more physical locations in the FPGA to reduce register-to-register delays. You may also want to choose **Speed** for the optimization method, which typically results in higher logic utilization due to logic duplication. By making use of the registers or FIFOs available in the bridges, you can increase the design speed and avoid needless logic duplication or speed optimizations, thereby reducing the logic utilization of the design.

12.5.2.2 Limiting Concurrency

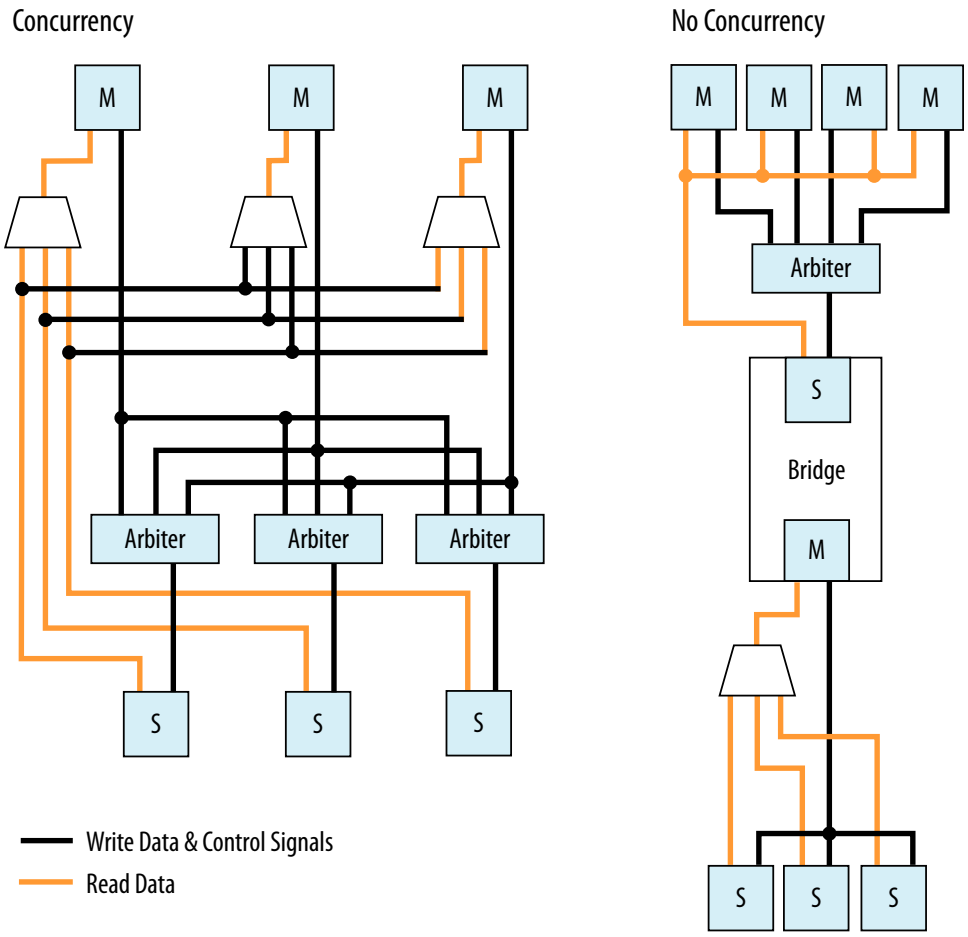
The amount of logic generated for the interconnect often increases as the system becomes larger because Qsys Pro creates arbitration logic for every slave interface that is shared by multiple master interfaces. Qsys Pro inserts multiplexer logic between master interfaces that connect to multiple slave interfaces if both support read datapaths.

Most embedded processor designs contain components that are either incapable of supporting high data throughput, or do not need to be accessed frequently. These components can contain master or slave interfaces. Because the interconnect supports concurrent accesses, you may want to limit concurrency by inserting bridges into the datapath to limit the amount of arbitration and multiplexer logic generated.

For example, if a system contains three master and three slave interfaces that are interconnected, Qsys Pro generates three arbiters and three multiplexers for the read datapath. If these masters do not require a significant amount of simultaneous throughput, you can reduce the resources that your design consumes by connecting the three masters to a pipeline bridge. The bridge controls the three slave interfaces and reduces the interconnect into a bus structure. Qsys Pro creates one arbitration block between the bridge and the three masters, and a single read datapath multiplexer between the bridge and three slaves, and prevents concurrency. This implementation is similar to a standard bus architecture.

You should not use this method for high throughput datapaths to ensure that you do not limit overall system performance.

Figure 219. Differences Between Systems With and Without a Pipeline Bridge



12.5.3 Using Bridges to Minimize Adapter Logic

Qsys Pro generates adapter logic for clock crossing, width adaptation, and burst support when there is a mismatch between the clock domains, widths, or bursting capabilities of the master and slave interface pairs.

Qsys Pro creates burst adapters when the maximum burst length of the master is greater than the master burst length of the slave. The adapter logic creates extra logic resources, which can be substantial when your system contains master interfaces connected to many components that do not share the same characteristics. By placing bridges in your design, you can reduce the amount of adapter logic that Qsys Pro generates.

12.5.3.1 Determining Effective Placement of Bridges

To determine the effective placement of a bridge, you should initially analyze each master in your system to determine if the connected slave devices support different bursting capabilities or operate in a different clock domain. The maximum burstcount of a component is visible as the `burstcount` signal in the HDL file of the component.

The maximum burst length is 2 $^{(width(burstcount\ -1))}$, therefore, if the `burstcount` width is four bits, the maximum burst length is eight. If no `burstcount` signal is present, the component does not support bursting or has a burst length of 1.

To determine if the system requires a clock crossing adapter between the master and slave interfaces, check the **Clock** column for the master and slave interfaces. If the clock is different for the master and slave interfaces, Qsys Pro inserts a clock crossing adapter between them. To avoid creating multiple adapters, you can place the components containing slave interfaces behind a bridge so that Qsys Pro creates a single adapter. By placing multiple components with the same burst or clock characteristics behind a bridge, you limit concurrency and the number of adapters.

You can also use a bridge to separate AXI and Avalon domains to minimize burst adaptation logic. For example, if there are multiple Avalon slaves that are connected to an AXI master, you can consider inserting a bridge to access the adaptation logic once before the bridge, instead of once per slave. This implementation results in latency, and you would also lose concurrency between reads and writes.

12.5.3.2 Changing the Response Buffer Depth

When you use automatic clock-crossing adapters, Qsys Pro determines the required depth of FIFO buffering based on the slave properties. If a slave has a high **Maximum Pending Reads** parameter, the resulting deep response buffer FIFO that Qsys Pro inserts between the master and slave can consume a lot of device resources. To control the response FIFO depth, you can use a clock crossing bridge and manually adjust its FIFO depth to trade off throughput with smaller memory utilization.

For example, if you have masters that cannot saturate the slave, you do not need response buffering. Using a bridge reduces the FIFO memory depth and reduces the **Maximum Pending Reads** available from the slave.

12.5.4 Considering the Effects of Using Bridges

Before you use pipeline or clock crossing bridges in a design, you should carefully consider their effects. Bridges can have any combination of consequences on your design, which could be positive or negative. Benchmarking your system before and after inserting bridges can help you determine the impact to the design.

12.5.4.1 Increased Latency

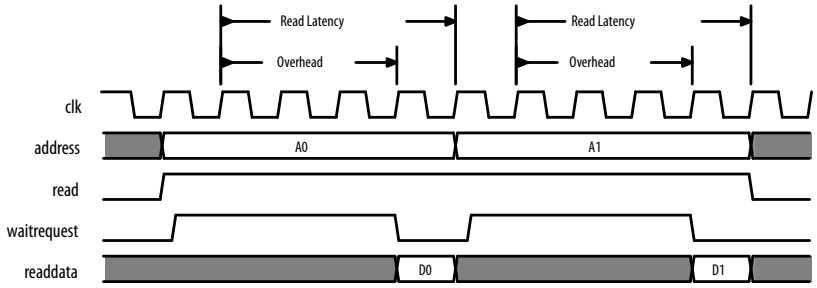
Adding a bridge to a design has an effect on the read latency between the master and the slave. Depending on the system requirements and the type of master and slave, this latency increase may not be acceptable in your design.

12.5.4.1.1 Acceptable Latency Increase

For a pipeline bridge, Qsys Pro adds a cycle of latency for each pipeline option that is enabled. The buffering in the clock crossing bridge also adds latency. If you use a pipelined or burst master that posts many read transfers, the increase in latency does not impact performance significantly because the latency increase is very small compared to the length of the data transfer.

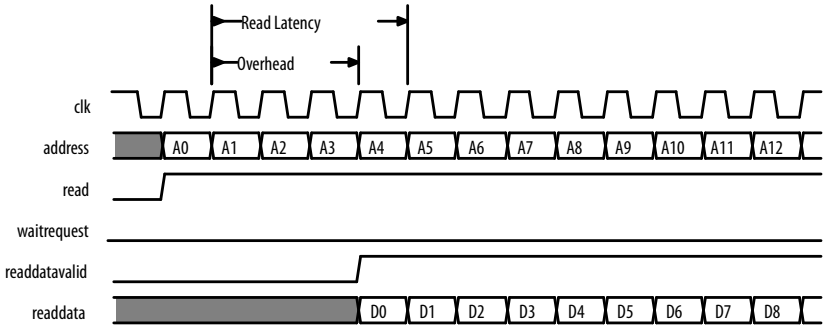
For example, if you use a pipelined read master such as a DMA controller to read data from a component with a fixed read latency of four clock cycles, but only perform a single word transfer, the overhead is three clock cycles out of the total of four. This is true when there is no additional pipeline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

Figure 220. Low-Efficiency Read Transfer



However, if 100 words of data are transferred without interruptions, the overhead is three cycles out of the total of 103 clock cycles. This corresponds to a read efficiency of approximately 97% when there is no additional pipeline latency in the interconnect. Adding a pipeline bridge to this read path adds two extra clock cycles of latency. The transfer requires 105 cycles to complete, corresponding to an efficiency of approximately 94%. Although the efficiency decreased by 3%, adding the bridge may increase the f_{MAX} by 5%. For example, if the clock frequency can be increased, the overall throughput would improve. As the number of words transferred increases, the efficiency increases to nearly 100%, whether or not a pipeline bridge is present.

Figure 221. High Efficiency Read Transfer

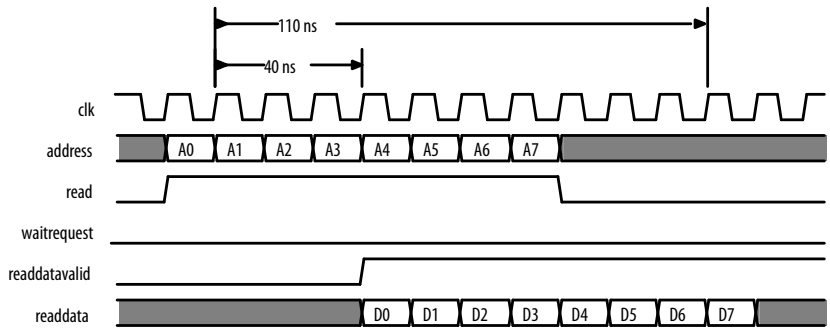


12.5.4.1.2 Unacceptable Latency Increase

Processors are sensitive to high latency read times and typically retrieve data for use in calculations that cannot proceed until the data arrives. Before adding a bridge to the datapath of a processor instruction or data master, determine whether the clock frequency increase justifies the added latency.

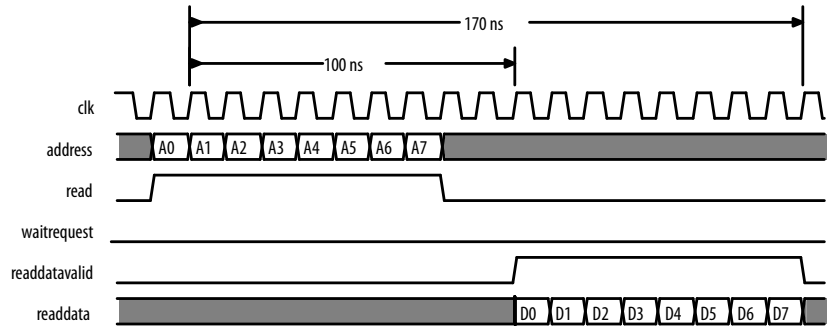
A Nios II processor instruction master has a cache memory with a read latency of four cycles, which is eight sequential words of data return for each read. At 100 MHz, the first read takes 40 ns to complete. Each successive word takes 10 ns so that eight reads complete in 110 ns.

Figure 222. Performance of a Nios II Processor and Memory Operating at 100 MHz



Adding a clock crossing bridge allows the memory to operate at 125 MHz. However, this increase in frequency is negated by the increase in latency because if the clock crossing bridge adds six clock cycles of latency at 100 MHz, then the memory continues to operate with a read latency of four clock cycles. Consequently, the first read from memory takes 100 ns, and each successive word takes 10 ns because reads arrive at the frequency of the processor, which is 100 MHz. In total, eight reads complete after 170 ns. Although the memory operates at a higher clock frequency, the frequency at which the master operates limits the throughput.

Figure 223. Performance of a Nios II Processor and Eight Reads with Ten Cycles Latency

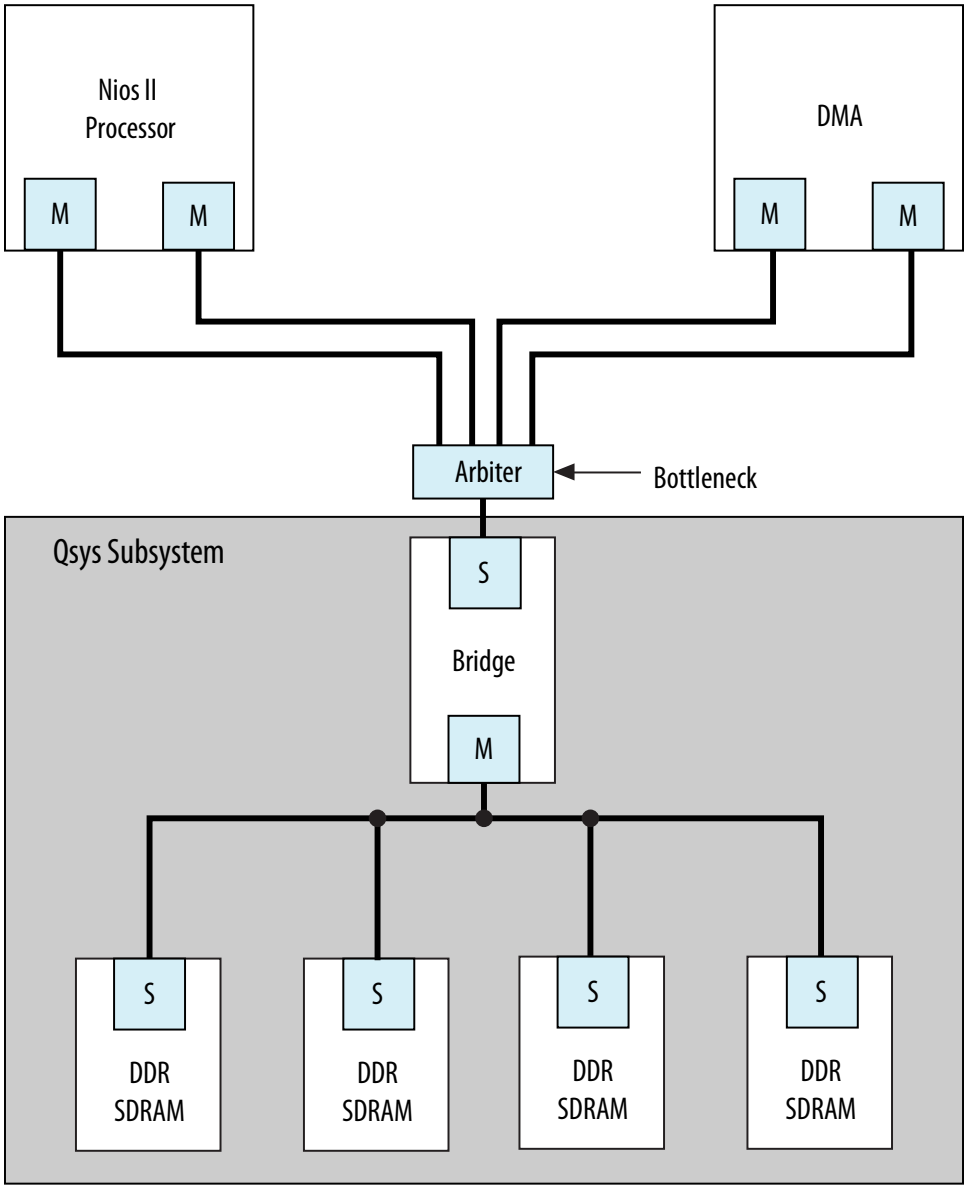


12.5.4.2 Limited Concurrency

Placing a bridge between multiple master and slave interfaces limits the number of concurrent transfers your system can initiate. This limitation is the same when connecting multiple master interfaces to a single slave interface. The slave interface of the bridge is shared by all the masters and, as a result, Qsys Pro creates arbitration logic. If the components placed behind a bridge are infrequently accessed, this concurrency limitation may be acceptable.

Bridges can have a negative impact on system performance if you use them inappropriately. For example, if multiple memories are used by several masters, you should not place the memory components behind a bridge. The bridge limits memory performance by preventing concurrent memory accesses. Placing multiple memory components behind a bridge can cause the separate slave interfaces to appear as one large memory to the masters accessing the bridge; all masters must access the same slave interface.

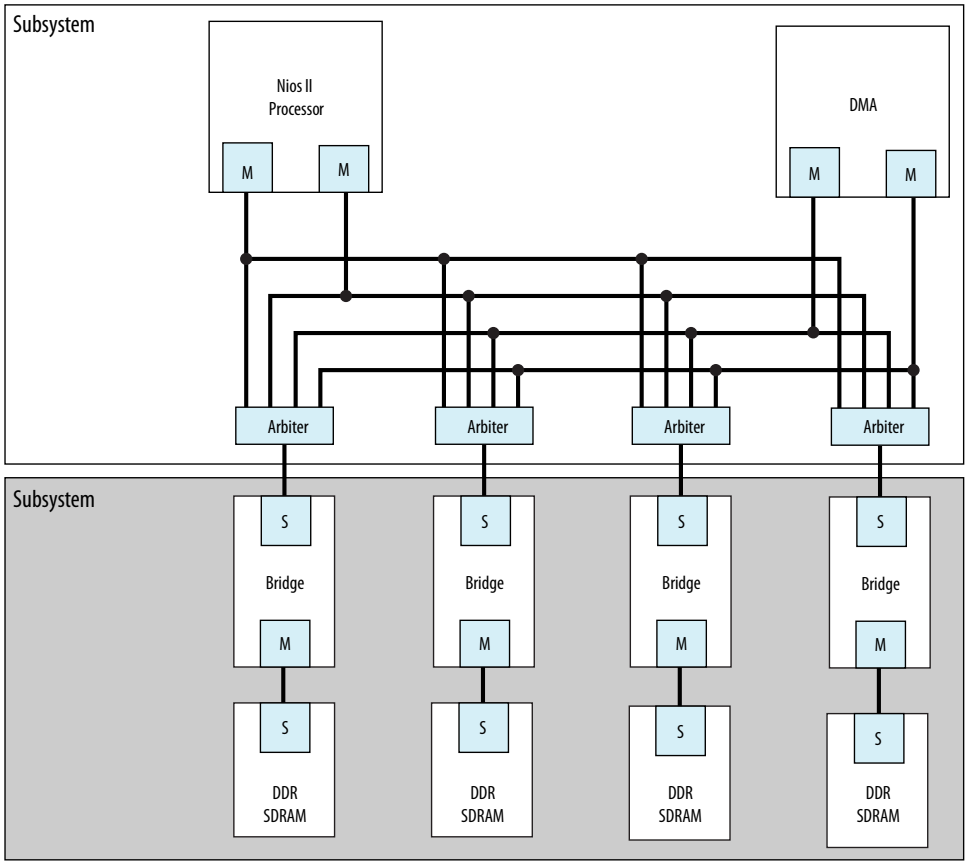
Figure 224. Inappropriate Use of a Bridge in a Hierarchical System



A memory subsystem with one bridge that acts as a single slave interface for the Avalon-MM Nios II and DMA masters, which results in a bottleneck architecture. The bridge acts as a bottleneck between the two masters and the memories.

If the f_{MAX} of your memory interfaces is low and you want to use a pipeline bridge between subsystems, you can place each memory behind its own bridge, which increases the f_{MAX} of the system without sacrificing concurrency.

Figure 225. Efficient Memory Pipelining Without a Bottleneck in a Hierarchical System

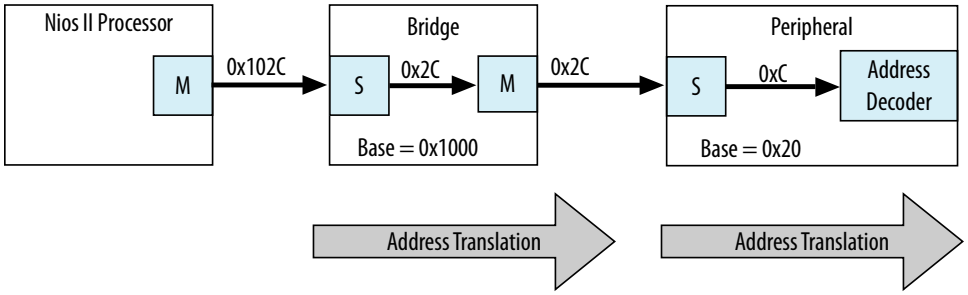


12.5.4.3 Address Space Translation

The slave interface of a pipeline or clock crossing bridge has a base address and address span. You can set the base address, or allow Qsys Pro to set it automatically. The address of the slave interface is the base offset address of all the components connected to the bridge. The address of components connected to the bridge is the sum of the base offset and the address of that component.

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

Figure 226. Bridge Address Translation

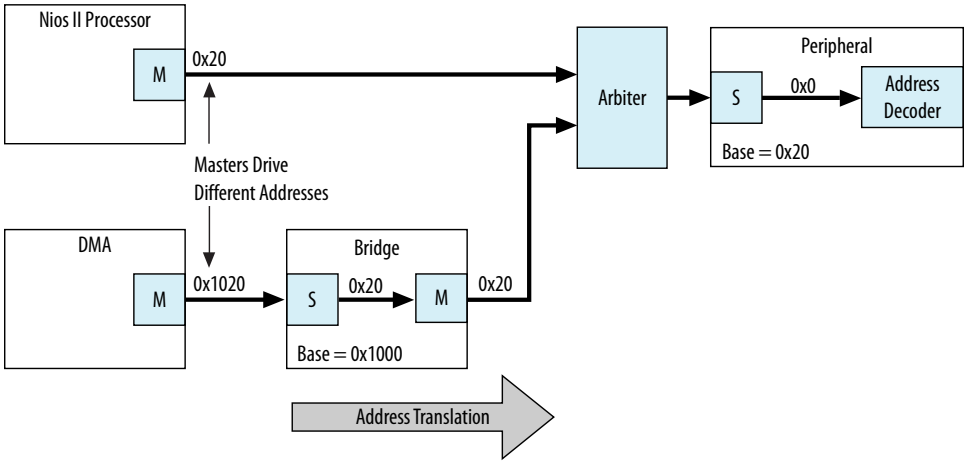


In this example, the Nios II processor connects to a bridge located at base address 0x1000, a slave connects to the bridge master interface at an offset of 0x20, and the processor performs a write transfer to the fourth 32-bit or 64-bit word within the slave. Nios II drives the address 0x102C to interconnect,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

12.5.4.4 Address Coherency

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

Figure 227. Slaves at Different Addresses and Complicating the System

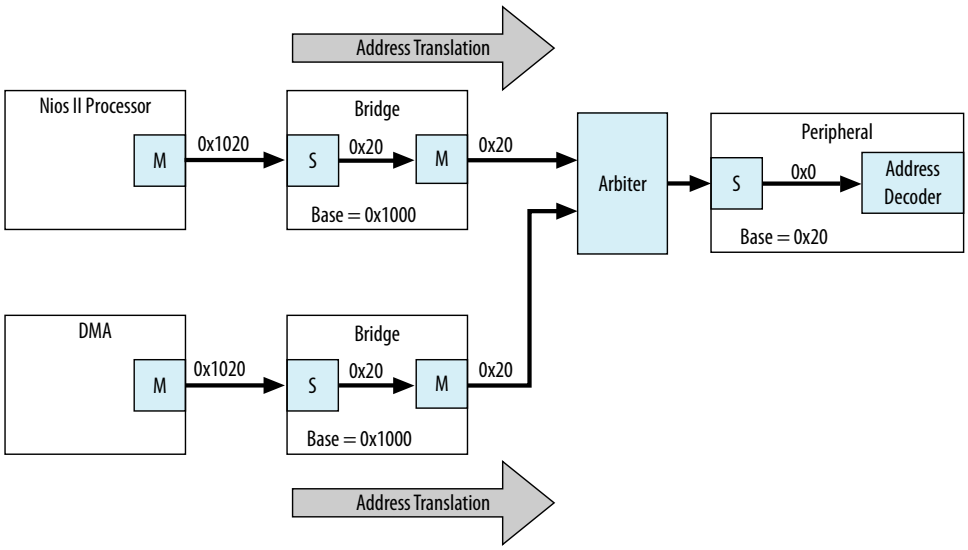


A Nios II processor and DMA controller access a slave interface located at address 0x20. The processor connects directly to the slave interface. The DMA controller connects to a pipeline bridge located at address 0x1000, which then connects to the slave interface. Because the DMA controller accesses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

To avoid the requirement for two addresses, you can add an additional bridge to the system, set its base address to 0x1000, and then disable all the pipelining options in the second bridge so that the bridge has minimal impact on system timing and

resource utilization. Because this second bridge has the same base address as the original bridge, the processor and DMA controller access the slave interface with the same address range.

Figure 228. Address Translation Corrected With Bridge



12.6 Increasing Transfer Throughput

Increasing the transfer efficiency of the master and slave interfaces in your system increases the throughput of your design. Designs with strict cost or power requirements benefit from increasing the transfer efficiency because you can then use less expensive, lower frequency devices. Designs requiring high performance also benefit from increased transfer efficiency because increased efficiency improves the performance of frequency–limited hardware.

Throughput is the number of symbols (such as bytes) of data that Qsys Pro can transfer in a given clock cycle. Read latency is the number of clock cycles between the address and data phase of a transaction. For example, a read latency of two means that the data is valid two cycles after the address is posted. If the master must wait for one request to finish before the next begins, such as with a processor, then the read latency is very important to the overall throughput.

You can measure throughput and latency in simulation by observing the waveforms, or using the verification IP monitors.

Related Links

- Avalon Verification IP Suite User Guide
- Mentor® Verification IP Altera Edition AMBA AXI3/4 User Guide

12.6.1 Using Pipelined Transfers

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

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

12.6.1.1 Using the Maximum Pending Reads Parameter

If you create a custom component with a slave interface supporting variable-latency reads, you must specify the **Maximum Pending Reads** parameter in the Component Editor. Qsys Pro uses this parameter to generate the appropriate interconnect and represent the maximum number of read transfers that your pipelined slave component can process. If the number of reads presented to the slave interface exceeds the **Maximum Pending Reads** parameter, then the slave interface must assert `waitrequest`.

Optimizing the value of the **Maximum Pending Reads** parameter requires an understanding of the latencies of your custom components. This parameter should be based on the component's highest read latency for the various logic paths inside the component. For example, if your pipelined component has two modes, one requiring two clock cycles and the other five, set the **Maximum Pending Reads** parameter to 5 to allow your component to pipeline five transfers, and eliminating dead cycles after the initial five-cycle latency.

You can also determine the correct value for the **Maximum Pending Reads** parameter by monitoring the number of reads that are pending during system simulation or while running the hardware. To use this method, set the parameter to a high value and use a master that issues read requests on every clock. You can use a DMA for this task as long as the data is written to a location that does not frequently assert `waitrequest`. If you implement this method, you can observe your component with a logic analyzer or built-in monitoring hardware.

Choosing the correct value for the **Maximum Pending Reads** parameter of your custom pipelined read component is important. If you underestimate the parameter value, you may cause a master interface to stall with a `waitrequest` until the slave responds to an earlier read request and frees a FIFO position.

The **Maximum Pending Reads** parameter controls the depth of the response FIFO inserted into the interconnect for each master connected to the slave. This FIFO does not use significant hardware resources. Overestimating the **Maximum Pending Reads** parameter results in a slight increase in hardware utilization. For these reasons, if you are not sure of the optimal value, you should overestimate this value.

If your system includes a bridge, you must set the **Maximum Pending Reads** parameter on the bridge as well. To allow maximum throughput, this value should be equal to or greater than the **Maximum Pending Reads** value for the connected slave that has the highest value. You can limit the maximum pending reads of a slave and

reduce the buffer depth by reducing the parameter value on the bridge if the high throughput is not required. If you do not know the **Maximum Pending Reads** value for all the slave components, you can monitor the number of reads that are pending during system simulation while running the hardware. To use this method, set the **Maximum Pending Reads** parameter to a high value and use a master that issues read requests on every clock, such as a DMA. Then, reduce the number of maximum pending reads of the bridge until the bridge reduces the performance of any masters accessing the bridge.

12.6.2 Arbitration Shares and Bursts

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

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

If a master cannot post a transfer and other masters are waiting to gain access to a particular slave, the arbiter grants another master access. This mechanism prevents a master from wasting arbitration cycles if it cannot post back-to-back transfers. A bursting transaction contains multiple beats (or words) of data, starting from a single address. Bursts allow a master to maintain access to a slave for more than a single word transfer. If a bursting master posts a write transfer with a burst length of eight, it is guaranteed arbitration for eight write cycles.

You can assign arbitration shares to an Avalon-MM bursting master and AXI masters (which are always considered a bursting master). Each share consists of one burst transaction (such as multi-cycle write), and allows a master to complete a number of bursts before arbitration switches to the next master.

12.6.2.1 Differences Between Arbitration Shares and Bursts

The following three key characteristics distinguish arbitration shares and bursts:

- Arbitration Lock
- Sequential Addressing
- Burst Adapters

Arbitration Lock

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

To avoid wasted bandwidth, your master designs should wait until a full burst transfer is ready before requesting access to a slave device. Alternatively, you can avoid wasted bandwidth by posting `burstcounts` equal to the amount of data that is ready. For example, if you create a custom bursting write master with a maximum `burstcount` of eight, but only three words of data are ready, you can present a

burstcount of three. This strategy does not result in optimal use of the system band width if the slave is capable of handling a larger burst; however, this strategy prevents stalling and allows access for other masters in the system.

Sequential Addressing

An Avalon-MM burst transfer includes a base address and a burstcount, which represents the number of words of data that are transferred, starting from the base address and incrementing sequentially. Burst transfers are common for processors, DMAs, and buffer processing accelerators; however, sometimes a master must access non-sequential addresses. Consequently, a bursting master must set the burstcount to the number of sequential addresses, and then reset the burstcount for the next location.

The arbitration share algorithm has no restrictions on addresses; therefore, your custom master can update the address it presents to the interconnect for every read or write transaction.

Burst Adapters

Qsys Pro allows you to create systems that mix bursting and non-bursting master and slave interfaces. This design strategy allows you to connect bursting master and slave interfaces that support different maximum burst lengths, with Qsys Pro generating burst adapters when appropriate.

Qsys Pro inserts a burst adapter whenever a master interface burst length exceeds the burst length of the slave interface, or if the master issues a burst type that the slave cannot support. For example, if you connect an AXI master to an Avalon slave, a burst adapter is inserted. Qsys Pro assigns non-bursting masters and slave interfaces a burst length of one. The burst adapter divides long bursts into shorter bursts. As a result, the burst adapter adds logic to the address and burstcount paths between the master and slave interfaces.

12.6.2.2 Choosing Avalon-MM Interface Types

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

12.6.2.2.1 Simple Avalon-MM Interfaces

Simple interface transfers do not support pipelining or bursting for reads or writes; consequently, their performance is limited. Simple interfaces are appropriate for transfers between masters and infrequently used slave interfaces. In Qsys Pro, the PIO, UART, and Timer include slave interfaces that use simple transfers.

12.6.2.2.2 Pipelined Avalon-MM Interfaces

Pipelined read transfers allow a pipelined master interface to start multiple read transfers in succession without waiting for prior transfers to complete. Pipelined transfers allow master-slave pairs to achieve higher throughput, even though the slave port may require one or more cycles of latency to return data for each transfer.

In many systems, read throughput becomes inadequate if simple reads are used and pipelined transfers can increase throughput. If you define a component with a fixed read latency, Qsys Pro automatically provides the pipelining logic necessary to support pipelined reads. You can use fixed latency pipelining as the default design starting

point for slave interfaces. If your slave interface has a variable latency response time, use the `readdatavalid` signal to indicate when valid data is available. The interconnect implements read response FIFO buffering to handle the maximum number of pending read requests.

To use components that support pipelined read transfers, and to use a pipelined system interconnect efficiently, your system must contain pipelined masters. You can use pipelined masters as the default starting point for new master components. Use the `readdatavalid` signal for these master interfaces.

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

Table 177. Pipeline Latency in a Master-Slave Pair

Master	Slave	Pipeline Management Logic Structure
No pipeline	No pipeline	Qsys Pro interconnect does not instantiate logic to handle pipeline latency.
No pipeline	Pipelined with fixed or variable latency	Qsys Pro interconnect forces the master to wait through any slave-side latency cycles. This master-slave pair gains no benefits from pipelining, because the master waits for each transfer to complete before beginning a new transfer. However, while the master is waiting, the slave can accept transfers from a different master.
Pipelined	No pipeline	Qsys Pro interconnect carries out the transfer as if neither master nor slave were pipelined, causing the master to wait until the slave returns data. An example of a non-pipeline slave is an asynchronous off-chip interface.
Pipelined	Pipelined with fixed latency	Qsys Pro interconnect allows the master to capture data at the exact clock cycle when data from the slave is valid, to enable maximum throughput. An example of a fixed latency slave is an on-chip memory.
Pipelined	Pipelined with variable latency	The slave asserts a signal when its `readdata` is valid, and the master captures the data. The master-slave pair can achieve maximum throughput if the slave has variable latency. Examples of variable latency slaves include SDRAM and FIFO memories.

12.6.2.2.3 Burst Avalon-MM Interfaces

Burst transfers are commonly used for latent memories such as SDRAM and off-chip communication interfaces, such as PCI Express. To use a burst-capable slave interface efficiently, you must connect to a bursting master. Components that require bursting to operate efficiently typically have an overhead penalty associated with short bursts or non-bursting transfers.

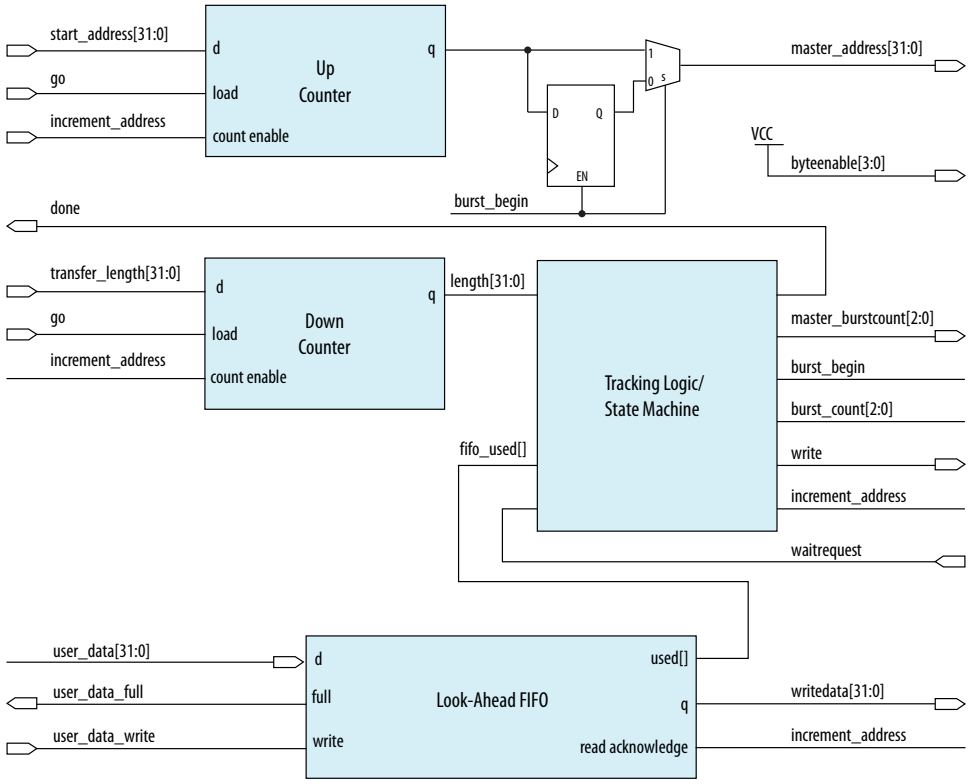
You can use a burst-capable slave interface if you know that your component requires sequential transfers to operate efficiently. Because SDRAM memories incur a penalty when switching banks or rows, performance improves when SDRAM memories are accessed sequentially with bursts.

Architectures that use the same signals to transfer address and data also benefit from bursting. Whenever an address is transferred over shared address and data signals, the throughput of the data transfer is reduced. Because the address phase adds overhead, using large bursts increases the throughput of the connection.

12.6.2.3 Avalon-MM Burst Master Example

Figure 229. Avalon Bursting Write Master

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



The master performs word accesses and writes to sequential memory locations. When `go` is asserted, the `start_address` and `transfer_length` are registered. On the next clock cycle, the control logic asserts `burst_begin`, which synchronizes the internal control signals in addition to the `master_address` and `master_burstcount` presented to the interconnect. The timing of these two signals is important because during bursting write transfers `byteenable` and `burstcount` must be held constant for the entire burst.

To avoid inefficient writes, the master posts a burst when enough data is buffered in the FIFO. To maximize the burst efficiency, the master should stall only when a slave asserts `waitrequest`. In this example, the FIFO's used signal tracks the number of words of data that are stored in the FIFO and determines when enough data has been buffered.

The `address` register increments after every word transfer, and the `length` register decrements after every word transfer. The address remains constant throughout the burst. Because a transfer is not guaranteed to complete on burst boundaries, additional logic is necessary to recognize the completion of short bursts and complete the transfer.

Related Links

Avalon Memory-Mapped Master Templates

12.7 Reducing Logic Utilization

You can minimize logic size of Qsys Pro systems. Typically, there is a trade-off between logic utilization and performance. Reducing logic utilization applies to both Avalon and AXI interfaces.

12.7.1 Minimizing Interconnect Logic to Reduce Logic Unitization

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

Related Links

Limited Concurrency on page 733

12.7.1.1 Creating Dedicated Master and Slave Connections to Minimize Interconnect Logic

You can create a system where a master interface connects to a single slave interface. This configuration eliminates address decoding, arbitration, and return data multiplexing, which simplifies the interconnect. Dedicated master-to-slave connections attain the same clock frequencies as Avalon-ST connections.

Typically, these one-to-one connections include an Avalon memory-mapped bridge or hardware accelerator. For example, if you insert a pipeline bridge between a slave and all other master interfaces, the logic between the bridge master and slave interface is reduced to wires. If a hardware accelerator connects only to a dedicated memory, no system interconnect logic is generated between the master and slave pair.

12.7.1.2 Removing Unnecessary Connections to Minimize Interconnect Logic

The number of connections between master and slave interfaces affects the f_{MAX} of your system. Every master interface that you connect to a slave interface increases the width of the multiplexer width. As a multiplexer width increases, so does the logic depth and width that implements the multiplexer in the FPGA. To improve system performance, connect masters and slaves only when necessary.

When you connect a master interface to many slave interfaces, the multiplexer for the read data signal grows. Avalon typically uses a `readdata` signal. AXI read data signals add a response status and last indicator to the read response channel using `rdata`, `rresp`, and `rlast`. Additionally, bridges help control the depth of multiplexers.

Related Links

Implementing Command Pipelining (Master-to-Slave) on page 726
When multiple masters share a slave device, you can use command pipelining to improve performance.

12.7.1.3 Simplifying Address Decode Logic

If address code logic is in the critical path, you may be able to change the address map to simplify the decode logic. Experiment with different address maps, including a one-hot encoding, to see if results improve.

12.7.2 Minimizing Arbitration Logic by Consolidating Multiple Interfaces

As the number of components in a design increases, the amount of logic required to implement the interconnect also increases. The number of arbitration blocks increases for every slave interface that is shared by multiple master interfaces. The width of the read data multiplexer increases as the number of slave interfaces supporting read transfers increases on a per master interface basis. For these reasons, consider implementing multiple blocks of logic as a single interface to reduce interconnect logic utilization.

12.7.2.1 Logic Consolidation Trade-Offs

You should consider the following trade-offs before making modifications to your system or interfaces:

- Consider the impact on concurrency that results when you consolidate components. When a system has four master components and four slave interfaces, it can initiate four concurrent accesses. If you consolidate the four slave interfaces into a single interface, then the four masters must compete for access. Consequently, you should only combine low priority interfaces such as low speed parallel I/O devices if the combination does not impact the performance.
- Determine whether consolidation introduces new decode and multiplexing logic for the slave interface that the interconnect previously included. If an interface contains multiple read and write address locations, the interface already contains the necessary decode and multiplexing logic. When you consolidate interfaces, you typically reuse the decoder and multiplexer blocks already present in one of the original interfaces; however, combining interfaces may simply move the decode and multiplexer logic, rather than eliminate duplication.
- Consider whether consolidating interfaces makes the design complicated. If so, you should not consolidate interfaces.

Related Links

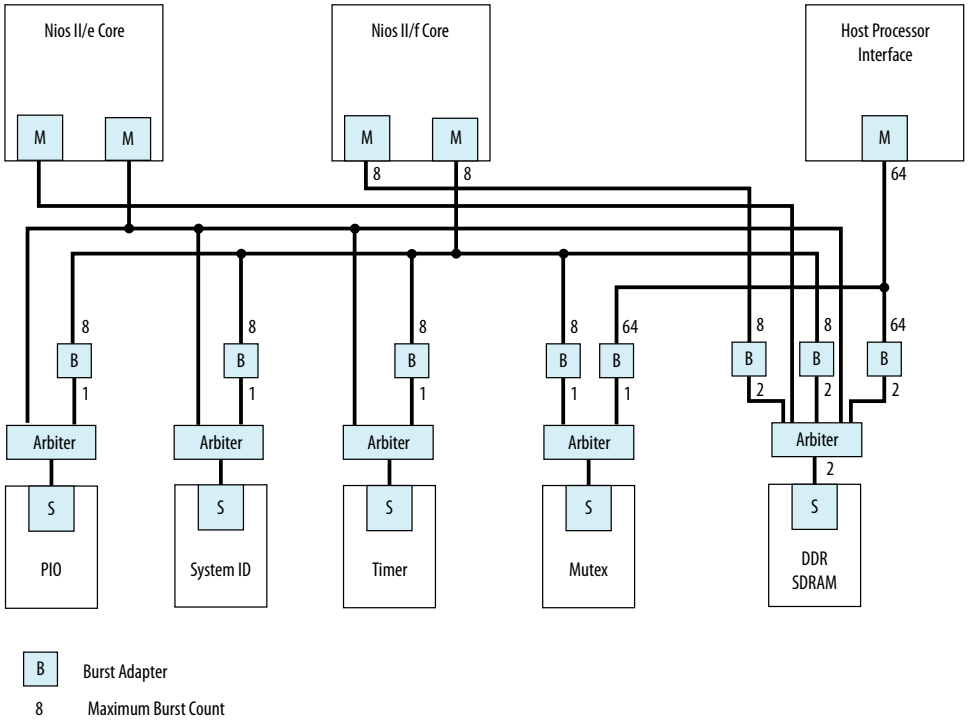
Using Concurrency in Memory-Mapped Systems on page 718
Qsys Pro interconnect uses parallel hardware in FPGAs, which allows you to design concurrency into your system and process transactions simultaneously.

12.7.2.2 Consolidating Interfaces

In this example, we have a system with a mix of components, each having different burst capabilities: a Nios II/e core, a Nios II/f core, and an external processor, which off-loads some processing tasks to the Nios II/f core.

The Nios II/f core supports a maximum burst size of eight. The external processor interface supports a maximum burst length of 64. The Nios II/e core does not support bursting. The memory in the system is SDRAM with an Avalon maximum burst length of two.

Figure 230. Mixed Bursting System

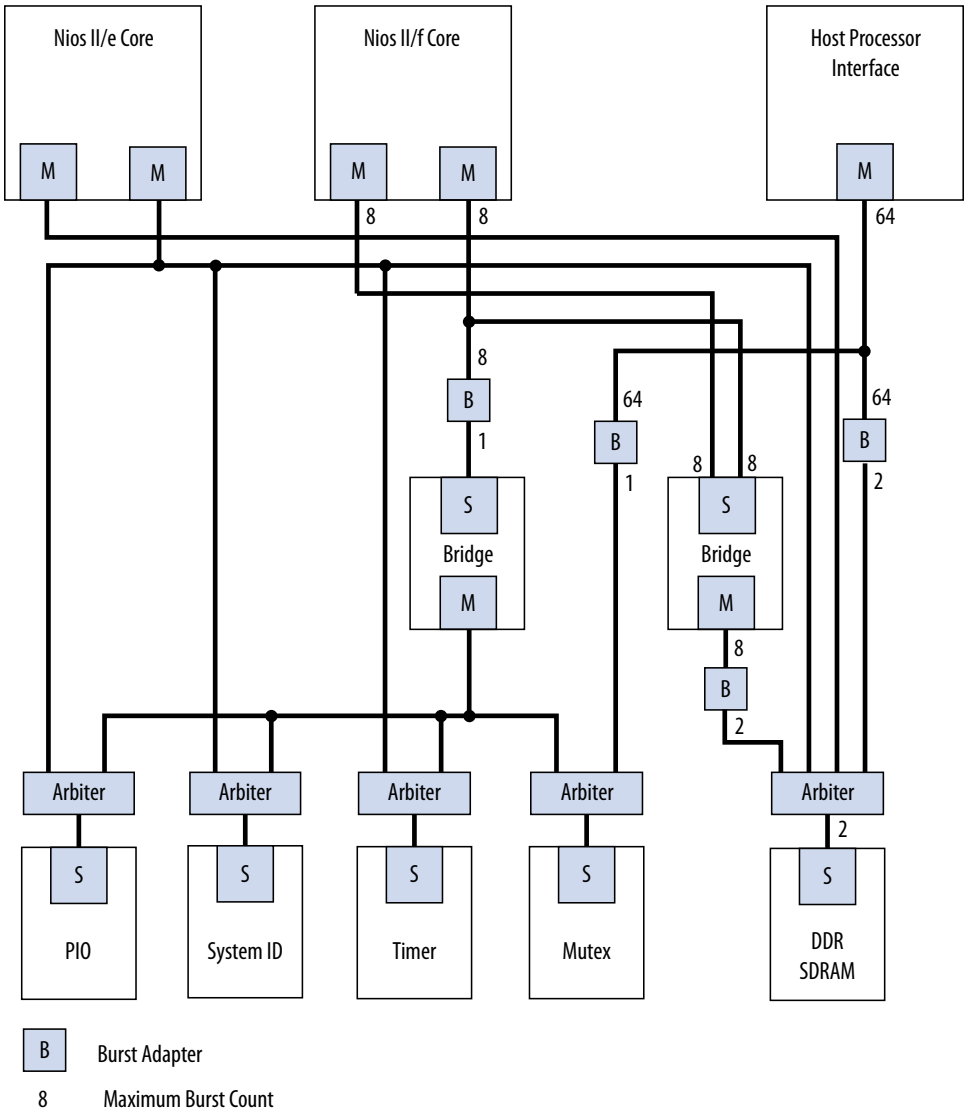


Qsys Pro automatically inserts burst adapters to compensate for burst length mismatches. The adapters reduce bursts to a single transfer, or the length of two transfers. For the external processor interface connecting to DDR SDRAM, a burst of 64 words is divided into 32 burst transfers, each with a burst length of two. When you generate a system, Qsys Pro inserts burst adapters based on maximum `burstcount` values; consequently, the interconnect logic includes burst adapters between masters and slave pairs that do not require bursting, if the master is capable of bursts.

In this example, Qsys Pro inserts a burst adapter between the Nios II processors and the timer, system ID, and PIO peripherals. These components do not support bursting and the Nios II processor performs a single word read and write accesses to these components.

Figure 231. Mixed Bursting System with Bridges

To reduce the number of adapters, you can add pipeline bridges. The pipeline bridge, between the Nios II/f core and the peripherals that do not support bursts, eliminates three burst adapters from the previous example. A second pipeline bridge between the Nios II/f core and the DDR SDRAM, with its maximum burst size set to eight, eliminates another burst adapter, as shown below.



12.7.3 Reducing Logic Utilization With Multiple Clock Domains

You specify clock domains in Qsys Pro on the **System Contents** tab. Clock sources can be driven by external input signals to Qsys Pro, or by PLLs inside Qsys Pro. Clock domains are differentiated based on the name of the clock. You can create multiple asynchronous clocks with the same frequency.

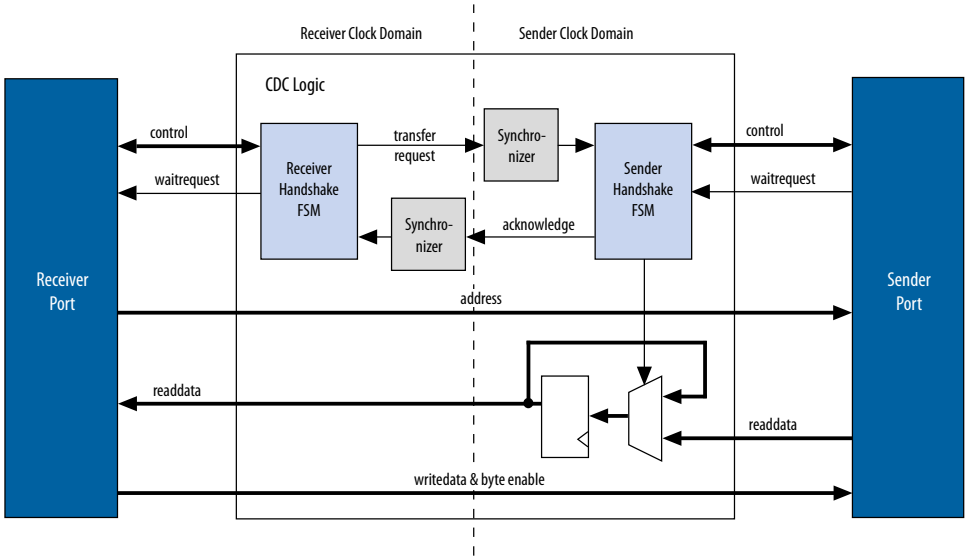
Qsys Pro generates Clock Domain Crossing (CDC) logic that hides the details of interfacing components operating in different clock domains. The interconnect supports the memory-mapped protocol with each port independently, and therefore masters do not need to incorporate clock adapters in order to interface to slaves on a different domain. Qsys Pro interconnect logic propagates transfers across clock domain boundaries automatically.

Clock-domain adapters provide the following benefits:

- Allows component interfaces to operate at different clock frequencies.
- Eliminates the need to design CDC hardware.
- Allows each memory-mapped port to operate in only one clock domain, which reduces design complexity of components.
- Enables masters to access any slave without communication with the slave clock domain.
- Allows you to focus performance optimization efforts on components that require fast clock speed.

A clock domain adapter consists of two finite state machines (FSM), one in each clock domain, that use a hand-shaking protocol to propagate transfer control signals (`read_request`, `write_request`, and the master `waitrequest` signals) across the clock boundary.

Figure 232. Clock Crossing Adapter



This example illustrates a clock domain adapter between one master and one slave. The synchronizer blocks use multiple stages of flipflops to eliminate the propagation of meta-stable events on the control signals that enter the handshake FSMs. The CDC logic works with any clock ratio.

The typical sequence of events for a transfer across the CDC logic is as follows:

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

Transfers proceed as normal on the slave and the master side, without a special protocol to handle crossing clock domains. From the perspective of a slave, there is nothing different about a transfer initiated by a master in a different clock domain. From the perspective of a master, a transfer across clock domains simply requires extra clock cycles. Similar to other transfer delay cases (for example, arbitration delay or wait states on the slave side), the Qsys Pro forces the master to wait until the transfer terminates. As a result, pipeline master ports do not benefit from pipelining when performing transfers to a different clock domain.

Qsys Pro automatically determines where to insert CDC logic based on the system and the connections between components, and places CDC logic to maintain the highest transfer rate for all components. Qsys Pro evaluates the need for CDC logic for each master and slave pair independently, and generates CDC logic wherever necessary.

Related Links

Avalon Memory-Mapped Design Optimizations

12.7.4 Duration of Transfers Crossing Clock Domains

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

- Four additional master clock cycles, due to the master-side clock synchronizer.
- Four additional slave clock cycles, due to the slave-side clock synchronizer.
- One additional clock in each direction, due to potential metastable events as the control signals cross clock domains.

Note: Systems that require a higher performance clock should use the Avalon-MM clock crossing bridge instead of the automatically inserted CDC logic. The clock crossing bridge includes a buffering mechanism so that multiple reads and writes can be pipelined. After paying the initial penalty for the first read or write, there is no additional latency penalty for pending reads and writes, increasing throughput by up to four times, at the expense of added logic resources.

12.8 Reducing Power Consumption

Qsys Pro provides various low power design changes that enable you to reduce the power consumption of the interconnect and custom components.

12.8.1 Reducing Power Consumption With Multiple Clock Domains

When you use multiple clock domains, you should put non-critical logic in the slower clock domain. Qsys Pro automatically reconciles data crossing over asynchronous clock domains by inserting clock crossing logic (handshake or FIFO).

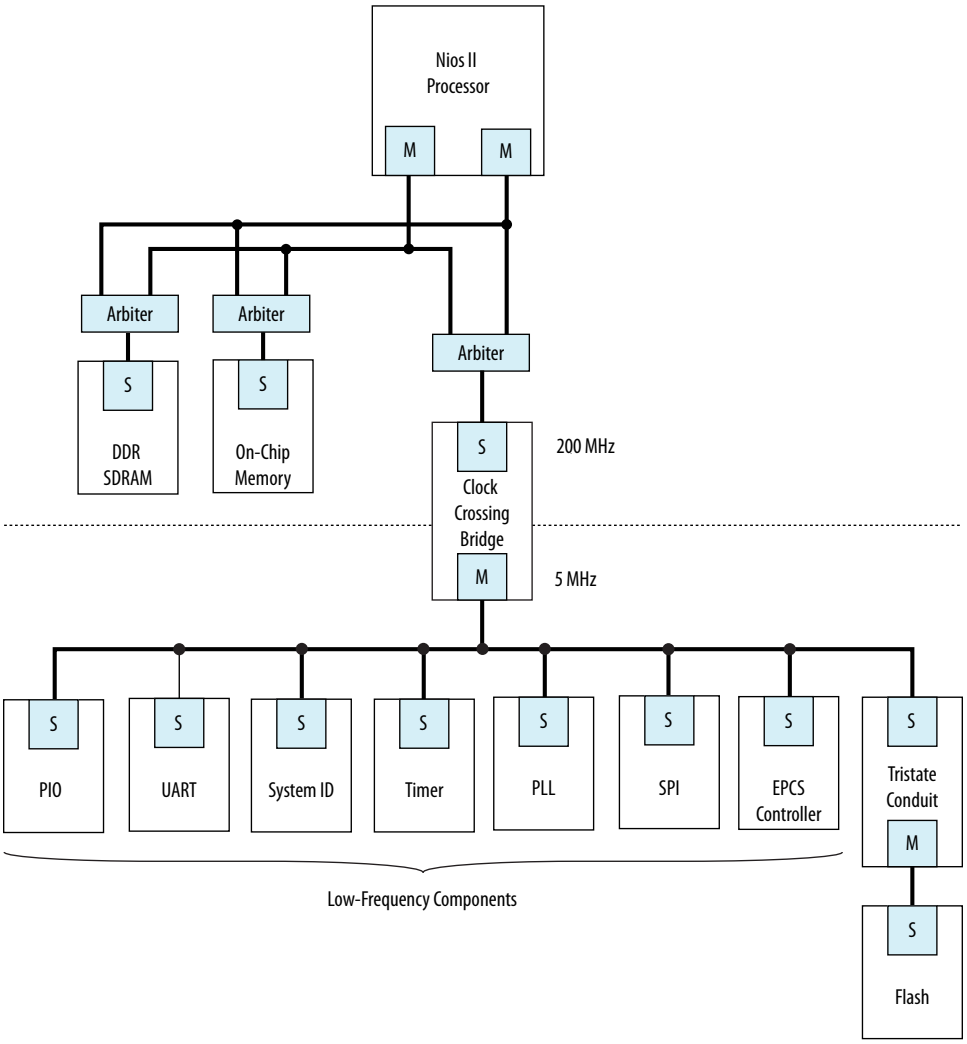
You can use clock crossing in Qsys Pro to reduce the clock frequency of the logic that does not require a high frequency clock, which allows you to reduce power consumption. You can use either handshaking clock crossing bridges or handshaking clock crossing adapters to separate clock domains.

You can use the clock crossing bridge to connect master interfaces operating at a higher frequency to slave interfaces running at a lower frequency. Only connect low throughput or low priority components to a clock crossing bridge that operates at a reduced clock frequency. The following are examples of low throughput or low priority components:

- PIOs
- UARTs (JTAG or RS-232)
- System identification (SysID)
- Timers
- PLL (instantiated within Qsys Pro)
- Serial peripheral interface (SPI)
- EPCS controller
- Tristate bridge and the components connected to the bridge

By reducing the clock frequency of the components connected to the bridge, you reduce the dynamic power consumption of the design. Dynamic power is a function of toggle rates and decreasing the clock frequency decreases the toggle rate.

Figure 233. Reducing Power Utilization Using a Bridge to Separate Clock Domains



Qsys Pro automatically inserts clock crossing adapters between master and slave interfaces that operate at different clock frequencies. You can choose the type of clock crossing adapter in the Qsys Pro **Project Settings** tab. Adapters do not appear in the **Connections** column because you do not insert them. The following clock crossing adapter types are available in Qsys Pro:

- **Handshake**—Uses a simple handshaking protocol to propagate transfer control signals and responses across the clock boundary. This adapter uses fewer hardware resources because each transfer is safely propagated to the target domain before the next transfer begins. The Handshake adapter is appropriate for systems with low throughput requirements.
- **FIFO**—Uses dual-clock FIFOs for synchronization. The latency of the FIFO adapter is approximately two clock cycles more than the handshake clock crossing component, but the FIFO-based adapter can sustain higher throughput because it supports multiple transactions simultaneously. The FIFO adapter requires more resources, and is appropriate for memory-mapped transfers requiring high throughput across clock domains.
- **Auto**—Qsys Pro specifies the appropriate FIFO adapter for bursting links and the Handshake adapter for all other links.

Because the clock crossing bridge uses FIFOs to implement the clock crossing logic, it buffers transfers and data. Clock crossing adapters are not pipelined, so that each transaction is blocking until the transaction completes. Blocking transactions may lower the throughput substantially; consequently, if you want to reduce power consumption without limiting the throughput significantly, you should use the clock crossing bridge or the FIFO clock crossing adapter. However, if the design requires single read transfers, a clock crossing adapter is preferable because the latency is lower.

The clock crossing bridge requires few logic resources other than on-chip memory. The number of on-chip memory blocks used is proportional to the address span, data width, buffering depth, and bursting capabilities of the bridge. The clock crossing adapter does not use on-chip memory and requires a moderate number of logic resources. The address span, data width, and the bursting capabilities of the clock crossing adapter determine the resource utilization of the device.

When you decide to use a clock crossing bridge or clock crossing adapter, you must consider the effects of throughput and memory utilization in the design. If on-chip memory resources are limited, you may be forced to choose the clock crossing adapter. Using the clock crossing bridge to reduce the power of a single component may not justify using more resources. However, if you can place all of the low priority components behind a single clock crossing bridge, you may reduce power consumption in the design.

Related Links

Power Optimization

12.8.2 Reducing Power Consumption by Minimizing Toggle Rates

A Qsys Pro system consumes power whenever logic transitions between on and off states. When the state is held constant between clock edges, no charging or discharging occurs. You can use the following design methodologies to reduce the toggle rates of your design:

- Registering component boundaries
- Using clock enable signals
- Inserting bridges

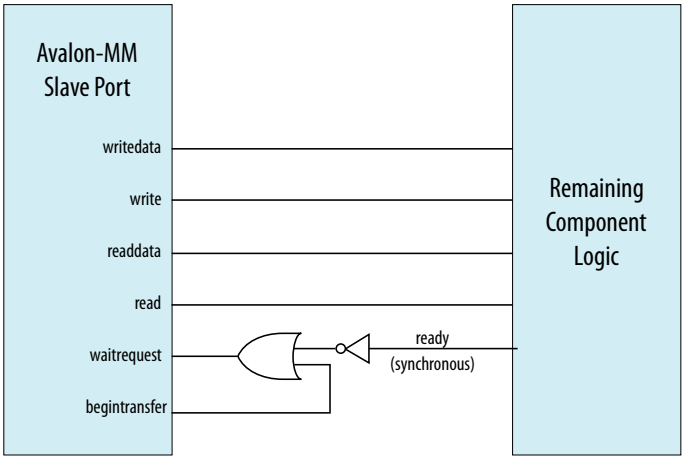
Qsys Pro interconnect is uniquely combinational when no adapters or bridges are present and there is no interconnect pipelining. When a slave interface is not selected by a master, various signals may toggle and propagate into the component. By registering the boundary of your component at the master or slave interface, you can minimize the toggling of the interconnect and your component. In addition, registering boundaries can improve operating frequency. When you register the signals at the interface level, you must ensure that the component continues to operate within the interface standard specification.

Avalon-MM `waitrequest` is a difficult signal to synchronize when you add registers to your component. The `waitrequest` signal must be asserted during the same clock cycle that a master asserts read or write to in order to prolong the transfer. A master interface can read the `waitrequest` signal too early and post more reads and writes prematurely.

Note: There is no direct AXI equivalent for `waitrequest` and `burstcount`, though the *AMBA Protocol Specification* implies that the AXI `ready` signal cannot depend combinatorially on the AXI `valid` signal. Therefore, Qsys Pro typically buffers AXI component boundaries for the `ready` signal.

For slave interfaces, the interconnect manages the `begintransfer` signal, which is asserted during the first clock cycle of any read or write transfer. If the `waitrequest` is one clock cycle late, you can logically OR the `waitrequest` and the `begintransfer` signals to form a new `waitrequest` signal that is properly synchronized. Alternatively, the component can assert `waitrequest` before it is selected, guaranteeing that the `waitrequest` is already asserted during the first clock cycle of a transfer.

Figure 234. Variable Latency



Using Clock Enables

You can use clock enables to hold the logic in a steady state, and the `write` and `read` signals as clock enables for slave components. Even if you add registers to your component boundaries, the interface can potentially toggle without the use of clock enables. You can also use the clock enable to disable combinational portions of the component.

For example, you can use an active high clock enable to mask the inputs into the combinational logic to prevent it from toggling when the component is inactive. Before preventing inactive logic from toggling, you must determine if the masking causes the circuit to function differently. If masking causes a functional failure, it may be possible to use a register stage to hold the combinational logic constant between clock cycles.

Inserting Bridges

You can use bridges to reduce toggle rates, if you do not want to modify the component by using boundary registers or clock enables. A bridge acts as a repeater where transfers to the slave interface are repeated on the master interface. If the bridge is not accessed, the components connected to its master interface are also not accessed. The master interface of the bridge remains idle until a master accesses the bridge slave interface.

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

12.8.3 Reducing Power Consumption by Disabling Logic

There are typically two types of low power modes: volatile and non-volatile. A volatile low power mode holds the component in a reset state. When the logic is reactivated, the previous operational state is lost. A non-volatile low power mode restores the previous operational state. You can use either software-controlled or hardware-controlled sleep modes to disable a component in order to reduce power consumption.

Software-Controlled Sleep Mode

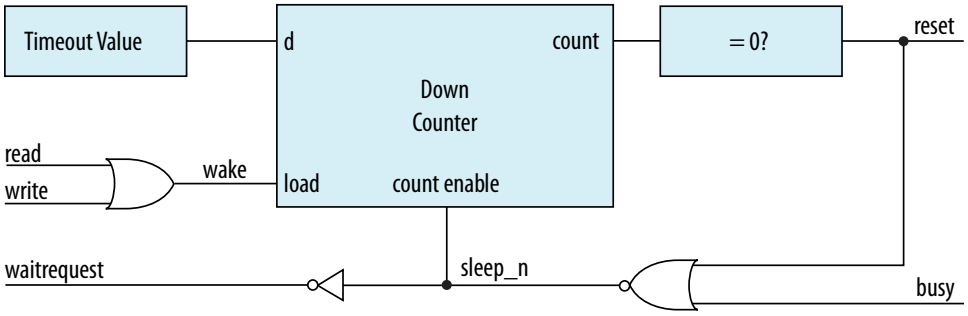
To design a component that supports software-controlled sleep mode, create a single memory-mapped location that enables and disables logic by writing a zero or one. You can use the register's output as a clock enable or reset, depending on whether the component has non-volatile requirements. The slave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

If multiple masters can access a component that supports sleep mode, you can use the mutex core to provide mutually exclusive accesses to your component. You can also build in the logic to re-enable the component on the very first access by any master in your system. If the component requires multiple clock cycles to re-activate, then it must assert a wait request to prolong the transfer as it exits sleep mode.

Hardware-Controlled Sleep Mode

Alternatively, you can implement a timer in your component that automatically causes the component to enter a sleep mode based on a timeout value specified in clock cycles between read or write accesses. Each access resets the timer to the timeout value. Each cycle with no accesses decrements the timeout value by one. If the counter reaches zero, the hardware enters sleep mode until the next access.

Figure 235. Hardware-Controlled Sleep Components



This example provides a schematic for the hardware-controlled sleep mode. If restoring the component to an active state takes a long time, use a long timeout value so that the component is not continuously entering and exiting sleep mode. The slave interface must remain functional while the rest of the component is in sleep mode. When the component exits sleep mode, the component must assert the `waitrequest` signal until it is ready for read or write accesses.

Related Links

Mutex Core

12.9 Reset Polarity and Synchronization in Qsys Pro

When you add a component interface with a reset signal, Qsys Pro defines its polarity as `reset`(active-high) or `reset_n` (active-low).

You can view the polarity status of a reset signal by selecting the signal in the **Hierarchy** tab, and then view its expanded definition in the open **Parameters** and **Block Symbol** tabs. When you generate your component, Qsys Pro interconnect automatically inverts ploarities as needed.

Figure 236. Reset Signal (Active-High)

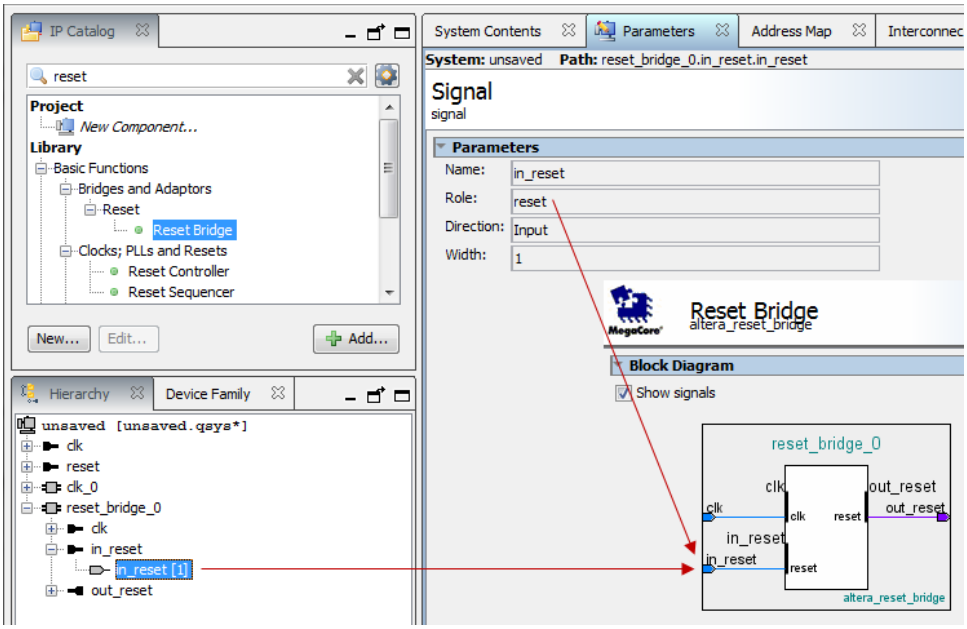
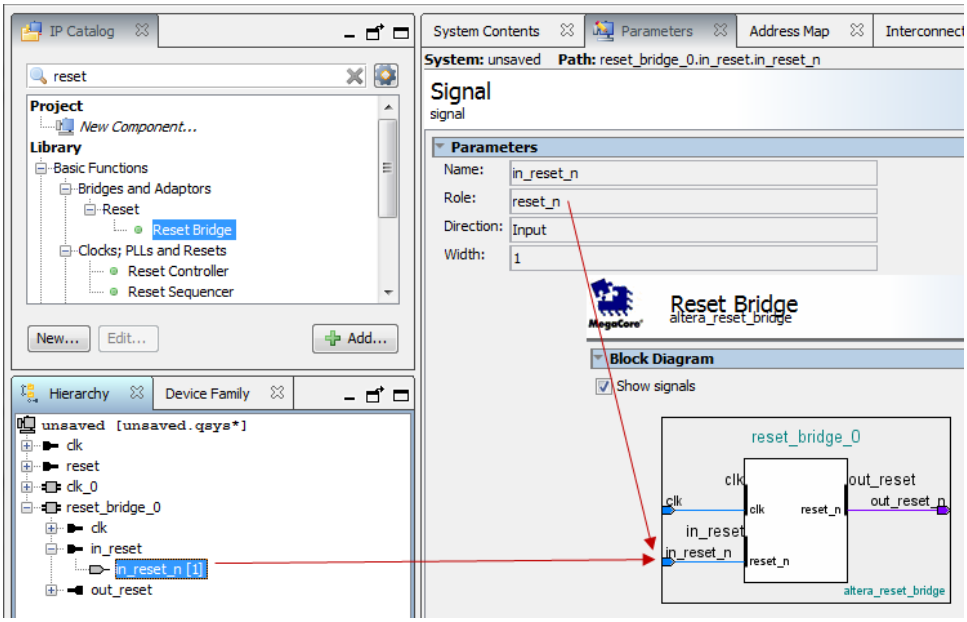


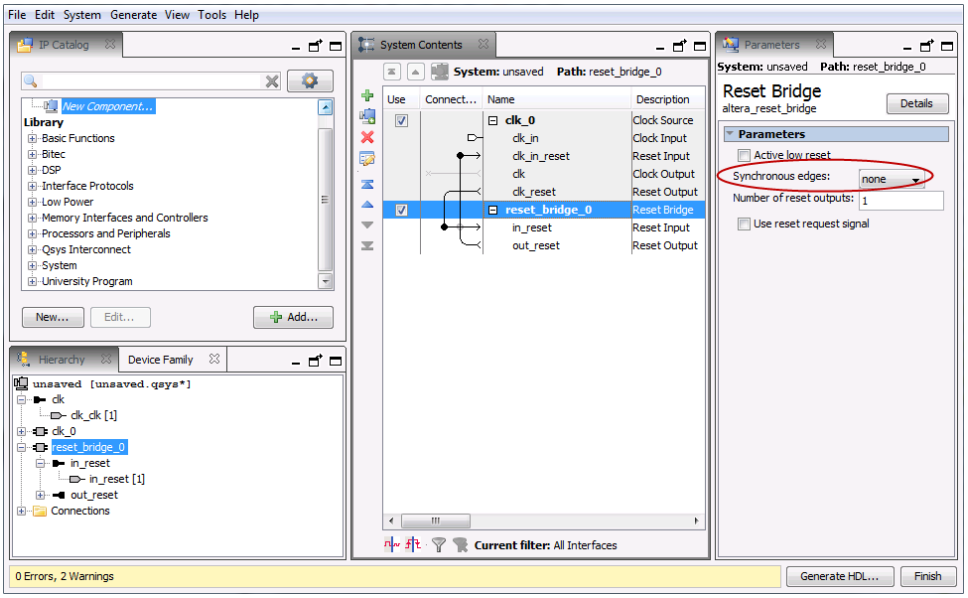
Figure 237. Reset Signal Active-Low



Each Qsys Pro component has its own requirements for reset synchronization. Some blocks have internal synchronization and have no requirements, whereas other blocks require an externally synchronized reset. You can define how resets are synchronized in your Qsys Pro system with the **Synchronous edges** parameter. In the clock source or reset bridge component, set the value of the **Synchronous edges** parameter to one of the following, depending on how the reset is externally synchronized:

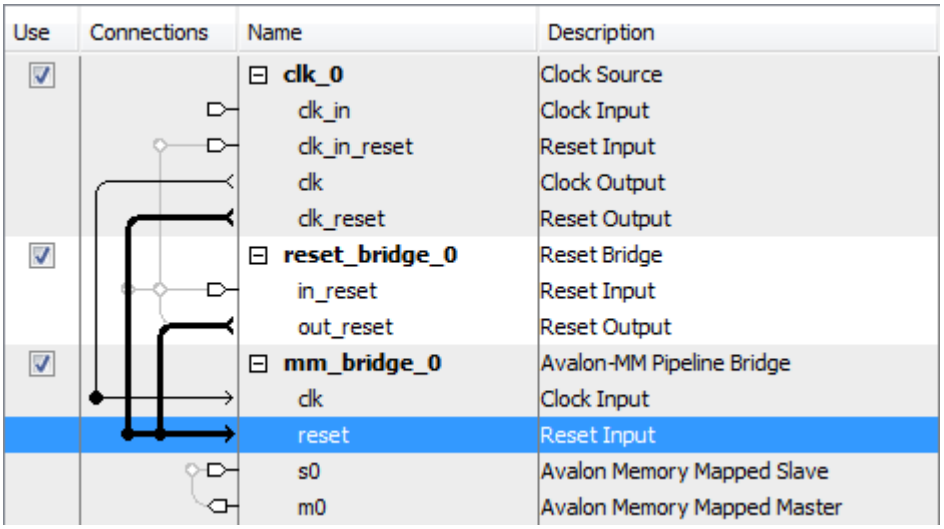
- **None**—There is no synchronization on this reset.
- **Both**—The reset is synchronously asserted and deasserted with respect to the input clock.
- **Deassert**—The reset is synchronously asserted with respect to the input clock, and asynchronously deasserted.

Figure 238. Synchronous Edges Parameter



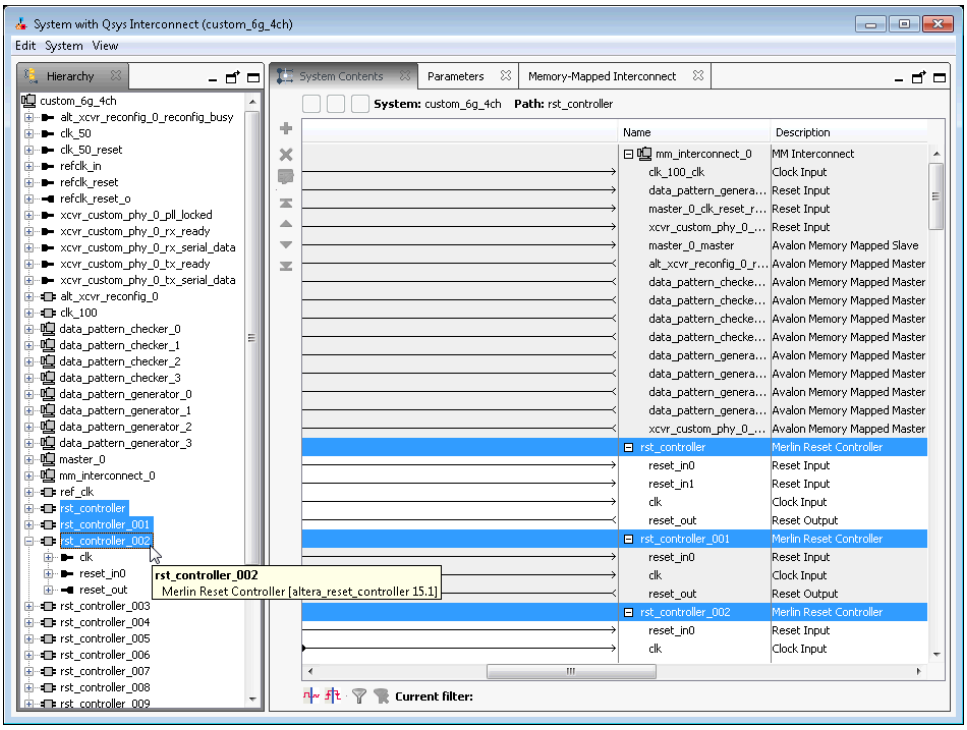
You can combine multiple reset sources to reset a particular component.

Figure 239. Combine Multiple Reset Sources



When you generate your component, Qsys Pro inserts adapters to synchronize or invert resets if there are mismatches in polarity or synchronization between the source and destination. You can view inserted adapters on the **Memory-Mapped Interconnect** tab with the **System ➤ Show System with Qsys Pro Interconnect** command.

Figure 240. Qsys Pro Interconnect



12.10 Optimizing Qsys Pro System Performance Design Examples

Avalon Pipelined Read Master Example on page 757

Multiplexer Examples on page 759

12.10.1 Avalon Pipelined Read Master Example

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

12.10.1.1 Avalon Pipelined Read Master Example Design Requirements

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

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

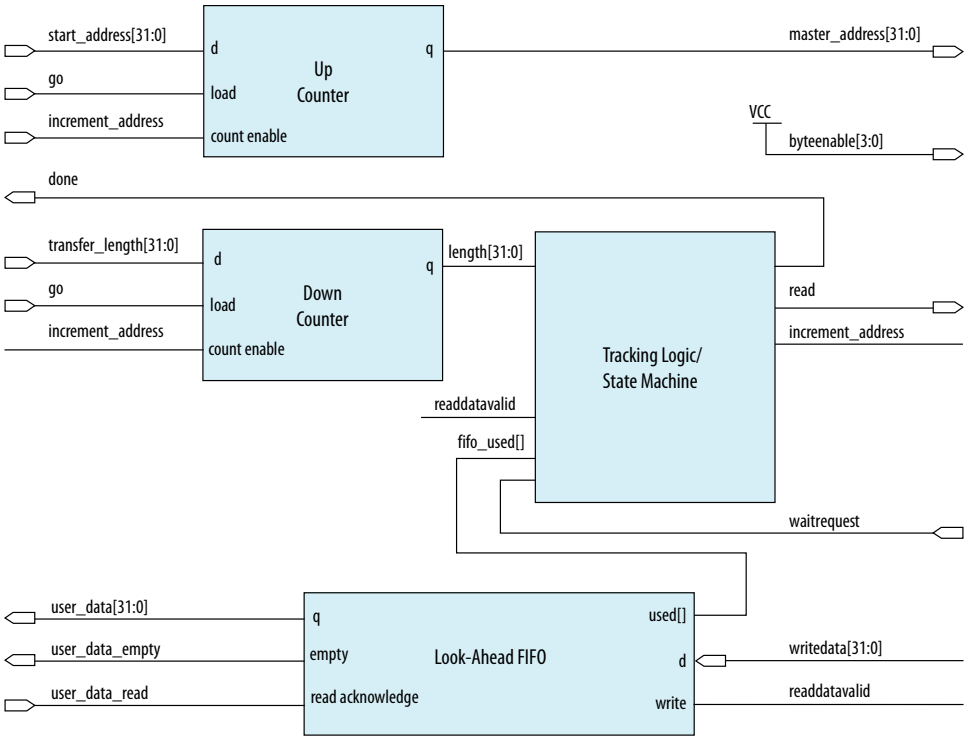
The datapath logic includes the `readdata` and `readdatavalid` signals. If your master can accept data on every clock cycle, you can register the data with the `readdatavalid` as an enable bit. If your master cannot process a continuous stream of read data, it must buffer the data in a FIFO. The control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

12.10.1.2 Expected Throughput Improvement

The throughput improvement that you can achieve with a pipelined read master is typically directly proportional to the pipeline depth of the interconnect and the slave interface. For example, if the total latency is two cycles, you can double the throughput by inserting a pipelined read master, assuming the slave interface also supports pipeline transfers. If either the master or slave does not support pipelined read transfers, then the interconnect asserts `waitrequest` until the transfer completes. You can also gain throughput when there are some cycles of overhead before a read response.

Where reads are not pipelined, the throughput is reduced. When both the master and slave interfaces support pipelined read transfers, data flows in a continuous stream after the initial latency. You can use a pipelined read master that stores data in a FIFO to implement a custom DMA, hardware accelerator, or off-chip communication interface.

Figure 241. Pipelined Read Master



This example shows a pipelined read master that stores data in a FIFO. The master performs word accesses that are word-aligned and reads from sequential memory addresses. The transfer length is a multiple of the word size.

When the go bit is asserted, the master registers the `start_address` and `transfer_length` signals. The master begins issuing reads continuously on the next clock cycle until the length register reaches zero. In this example, the word size is four bytes so that the address always increments by four, and the length decrements by four. The `read` signal remains asserted unless the FIFO fills to a predetermined level. The address register increments and the length register decrements if the length has not reached 0 and a read is posted.

The master posts a read transfer every time the `read` signal is asserted and the `waitrequest` is deasserted. The master issues reads until the entire buffer has been read or `waitrequest` is asserted. An optional tracking block monitors the done bit. When the length register reaches zero, some reads are outstanding. The tracking logic prevents assertion of done until the last read completes, and monitors the number of reads posted to the interconnect so that it does not exceed the space remaining in the `readdata` FIFO. This example includes a counter that verifies that the following conditions are met:

- If a read is posted and `readdatavalid` is deasserted, the counter increments.
- If a read is not posted and `readdatavalid` is asserted, the counter decrements.

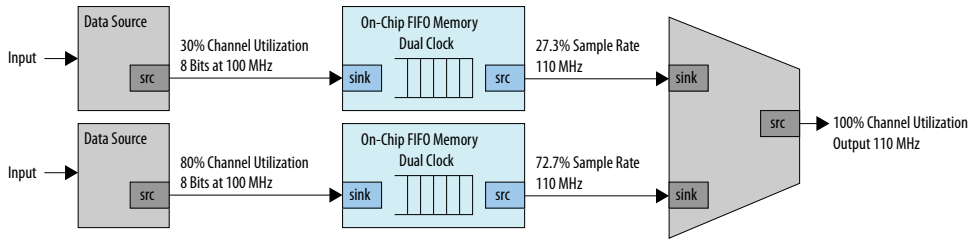
When the `length` register and the tracking logic counter reach zero, all the reads have completed and the done bit is asserted. The done bit is important if a second master overwrites the memory locations that the pipelined read master accesses. This bit guarantees that the reads have completed before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

12.10.2 Multiplexer Examples

You can combine adapters with streaming components to create datapaths whose input and output streams have different properties. The following examples demonstrate datapaths in which the output stream exhibits higher performance than the input stream.

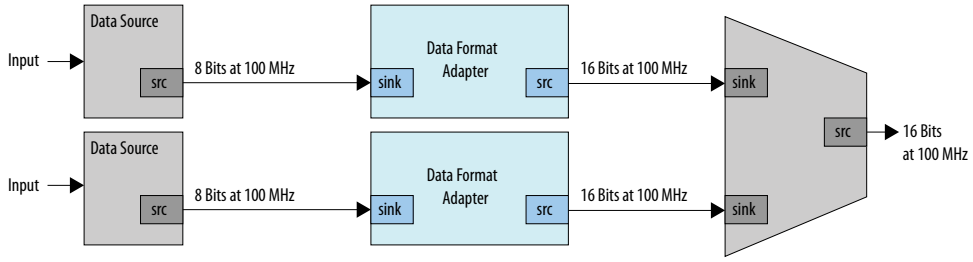
The diagram below illustrates a datapath that uses the dual clock version of the on-chip FIFO memory to boost the frequency of input data from 100 MHz to 110 MHz by sampling two input streams at differential rates. The on-chip FIFO memory has an input clock frequency of 100 MHz, and an output clock frequency of 110 MHz. The channel multiplexer runs at 110 MHz and samples one input stream 27.3 percent of the time, and the second 72.7 percent of the time. You must know what the typical and maximum input channel utilizations are before for this type of design. For example, if the first channel hits 50% utilization, the output stream exceeds 100% utilization.

Figure 242. Datapath that Doubles the Clock Frequency



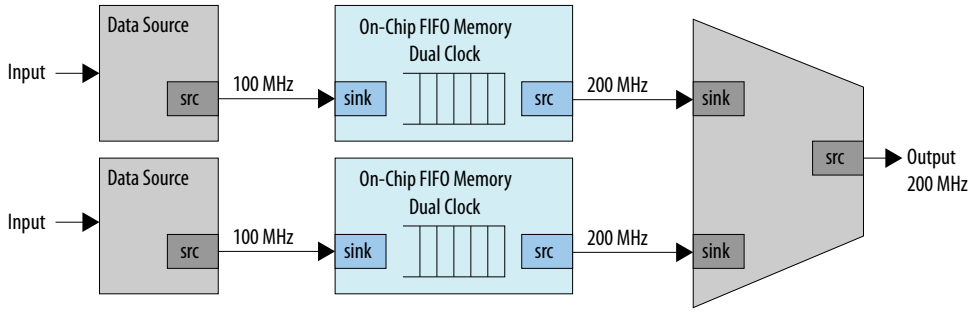
The diagram below illustrates a datapath that uses a data format adapter and Avalon-ST channel multiplexer to merge the 8-bit 100 MHz input from two streaming data sources into a single 16-bit 100 MHz streaming output. This example shows an output with double the throughput of each interface with a corresponding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

Figure 243. Datapath to Double Data Width and Maintain Original Frequency



The diagram below illustrates a datapath that uses the dual clock version of the on-chip FIFO memory and Avalon-ST channel multiplexer to merge the 100 MHz input from two streaming data sources into a single 200 MHz streaming output. This example shows an output with double the throughput of each interface with a corresponding doubling of the clock frequency.

Figure 244. Datapath to Boost the Clock Frequency



12.11 Document Revision History

The table below indicates edits made to the *Optimizing Qsys Pro System Performance* content since its creation.

Table 178. Document Revision History

Date	Version	Changes
2016.10.31	16.1.0	• Implemented Intel rebranding. • Implemented Qsys Pro rebranding.
2015.11.02	15.1.0	• Added:*Reset Polarity and Synchronization in Qsys Pro*. • Changed instances of *Quartus II* to *Quartus Prime*.
2015.05.04	15.0.0	*Multiplexer Examples*, rearranged description text for the figures.
May 2013	13.0.0	AMBA APB support.
November 2012	12.1.0	AMBA AXI4 support.
June 2012	12.0.0	AMBA AXI3 support.
November 2011	11.1.0	New document release.

Related Links

Altera Documentation Archive

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

13 Component Interface Tcl Reference

Tcl commands allow you to perform a wide range of functions in Qsys Pro. Command descriptions contain the Qsys Pro phases where you can use the command, for example, main program, elaboration, composition, or fileset callback.

Qsys Pro supports Avalon, AMBA AXI3 (version 1.0), AMBA AXI4 (version 2.0), AMBA AXI4-Lite (version 2.0), AMBA AXI4-Stream (version 1.0), and AMBA APB3 (version 1.0) interface specifications.

For more information about procedures for creating IP component `_hw.tcl` files in the Qsys Pro Component Editor, and supported interface standards, refer to *Creating Qsys Pro Components* and *Qsys Pro Interconnect* .

If you are developing an IP component to work with the Nios II processor, refer to *Publishing Component Information to Embedded Software* in section 3 of the *Nios II Software Developer's Handbook*, which describes how to publish hardware IP component information for embedded software tools, such as a C compiler and a Board Support Package (BSP) generator.

Related Links

- Avalon Interface Specifications
- AMBA Protocol Specifications
- Creating Qsys Pro Components on page 585
  You can create a Hardware Component Definition File (`_hw.tcl`) to describe and package IP components for use in a Qsys Pro system.
- Qsys Pro Interconnect on page 627
  Qsys Pro interconnect is a high-bandwidth structure that allows you to connect IP components to other IP components with various interfaces.
- Publishing Component Information to Embedded Software
  In *Nios II Gen2 Software Developer's Handbook*

13.1 Qsys Pro _hw.tcl Command Reference

13.1.1 Interfaces and Ports

13.1.1.1 add_interface

Description

Adds an interface to your module. An interface represents a collection of related signals that are managed together in the parent system. These signals are implemented in the IP component's HDL, or exported from an interface from a child instance. As the IP component author, you choose the name of the interface.

Availability

Discovery, Main Program, Elaboration, Composition

Usage

`add_interface` <*name*> <*type*> <*direction*> [<*associated_clock*>]

Returns

No returns value.

Arguments

name A name you choose to identify an interface.

type The type of interface.

direction The interface direction.

associated_clock (optional) (deprecated) For interfaces requiring associated clocks, use: `set_interface_property` <*interface*> `associatedClock` <*clockInterface*> For interfaces requiring associated resets, use: `set_interface_property` <*interface*> `associatedReset` <*resetInterface*>

Example

```
add_interface mm_slave avalon slave

add_interface my_export conduit end
set_interface_property my_export EXPORT_OF uart_0.external_connection
```

Notes

By default, interfaces are enabled. You can set the interface property ENABLED to `false` to disable an interface. If an interface is disabled, it is hidden and its ports are automatically terminated to their default values. Active high signals are terminated to 0. Active low signals are terminated to 1.

If the IP component is composed of child instances, the top-level interface is associated with a child instance's interface with `set_interface_property` *`interface`* `EXPORT_OF` *`child_instance.interface`*.

The following direction rules apply to Qsys Pro-supported interfaces.

Interface Type	Direction
avalon	master, slave
axi	master, slave
tristate_conduit	master, slave
avalon_streaming	source, sink
interrupt	sender, receiver
conduit	end
clock	source, sink
reset	source, sink
nios_custom_instruction	slave

Related Links

- add_interface_port on page 766
- get_interface_assignments on page 770
- get_interface_properties on page 772
- get_interfaces on page 768

13.1.1.2 add_interface_port

Description

Adds a port to an interface on your module. The name must match the name of a signal on the top-level module in the HDL of your IP component. The port width and direction must be set before the end of the elaboration phase. You can set the port width as follows:

- In the Main program, you can set the port width to a fixed value or a width expression.
- If the port width is set to a fixed value in the Main program, you can update the width in the elaboration callback.

Availability

Main Program, Elaboration

Usage

`add_interface_port` *<interface>* *<port>* [*<signal_type>* *<direction>* *<width_expression>*]

Returns

Arguments

interface The name of the interface to which this port belongs.

port The name of the port. This name must match a signal in your top-level HDL for this IP component.

signal_type (optional) The type of signal for this port, which must be unique. Refer to the *Avalon Interface Specifications* for the signal types available for each interface type.

direction (optional) The direction of the signal. Refer to *Direction Properties*.

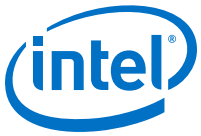
width_expression (optional) The width of the port, in bits. The width may be a fixed value, or a simple arithmetic expression of parameter values.

Example

```
fixed width:
add_interface_port mm_slave s0_rdata readdata output 32

width expression:
add_parameter DATA_WIDTH INTEGER 32
add_interface_port s0 rdata readdata output "DATA_WIDTH/2"
```

Related Links

- add_interface on page 764
- get_port_properties on page 774



- get_port_property on page 775
- get_port_property on page 775
- Direction Properties on page 858
- Avalon Interface Specifications

13.1.1.3 get_interfaces

Description

Returns a list of top-level interfaces.

Availability

Discovery, Main Program, Edit, Elaboration, Validation, Generation, Composition, Fileset Generation, Parameter Upgrade

Usage

```
get_interfaces
```

Returns

A list of the top-level interfaces exported from the system.

Arguments

No arguments.

Example

```
get_interfaces
```

Related Links

add_interface on page 764

13.1.1.4 get_interface_assignment

Description

Returns the value of the specified assignment for the specified interface

Availability

Main Program, Elaboration, Validation, Composition

Usage

`get_interface_assignment` <*interface*> <*assignment*>

Returns

The value of the assignment.

Arguments

interface The name of a top-level interface.

assignment The name of an assignment.

Example

```
get_interface_assignment s1 embeddedsw.configuration.isFlash
```

Related Links

- add_interface on page 764
- get_interface_assignments on page 770
- get_interfaces on page 768

13.1.1.5 get_interface_assignments

Description

Returns the value of all interface assignments for the specified interface.

Availability

Main Program, Elaboration, Validation, Composition

Usage

`get_interface_assignments` <*interface*>

Returns

A list of assignment keys.

Arguments

interface The name of the top-level interface whose assignment is being retrieved.

Example

```
get_interface_assignments s1
```

Related Links

- add_interface on page 764
- get_interface_assignment on page 769
- get_interfaces on page 768

13.1.1.6 get_interface_ports

Description

Returns the names of all of the ports that have been added to a given interface. If the interface name is omitted, all ports for all interfaces are returned.

Availability

Discovery, Main Program, Edit, Elaboration, Validation, Generation, Composition, Fileset Generation, Parameter Upgrade

Usage

`get_interface_ports` [<*interface*>]

Returns

A list of port names.

Arguments

interface (optional) The name of a top-level interface.

Example

```
get_interface_ports mm_slave
```

Related Links

- add_interface_port on page 766
- get_port_property on page 775
- set_port_property on page 779

13.1.1.7 get_interface_properties

Description

Returns the names of all the interface properties for the specified interface as a space separated list

Availability

Discovery, Main Program, Edit, Elaboration, Validation, Generation, Composition, Fileset Generation, Parameter Upgrade

Usage

`get_interface_properties` <*interface*>

Returns

A list of properties for the interface.

Arguments

interface The name of an interface.

Example

```
get_interface_properties interface
```

Notes

The properties for each interface type are different. Refer to the *Avalon Interface Specifications* for more information about interface properties.

Related Links

- get_interface_property on page 773
- set_interface_property on page 778
- Avalon Interface Specifications

13.1.1.8 get_interface_property

Description

Returns the value of a single interface property from the specified interface.

Availability

Discovery, Main Program, Elaboration, Composition, Fileset Generation

Usage

`get_interface_property` <*interface*> <*property*>

Returns

Arguments

interface The name of an interface.

property The name of the property whose value you want to retrieve. Refer to *Interface Properties*.

Example

```
get_interface_property mm_slave linewrapBursts
```

Notes

The properties for each interface type are different. Refer to the *Avalon Interface Specifications* for more information about interface properties.

Related Links

- get_interface_properties on page 772
- set_interface_property on page 778
- Avalon Interface Specifications

13.1.1.9 get_port_properties

Description

Returns a list of port properties.

Availability

Discovery, Main Program, Edit, Elaboration, Validation, Generation, Composition, Fileset Generation, Parameter Upgrade

Usage

```
get_port_properties
```

Returns

A list of port properties. Refer to *Port Properties*.

Arguments

No arguments.

Example

```
get_port_properties
```

Related Links

- add_interface_port on page 766
- get_port_property on page 775
- set_port_property on page 779
- Port Properties on page 857

13.1.1.10 get_port_property

Description

Returns the value of a property for the specified port.

Availability

Discovery, Main Program, Edit, Elaboration, Validation, Generation, Composition, Fileset Generation, Parameter Upgrade

Usage

`get_port_property` <*port*> <*property*>

Returns

The value of the property.

Arguments

port The name of the port.

property The name of a port property. Refer to *Port Properties*.

Example

```
get_port_property rdata WIDTH_VALUE
```

Related Links

- add_interface_port on page 766
- get_port_properties on page 774
- set_port_property on page 779
- Port Properties on page 857

13.1.1.11 set_interface_assignment

Description

Sets the value of the specified assignment for the specified interface.

Availability

Main Program, Elaboration, Validation, Composition

Usage

`set_interface_assignment` <*interface*> <*assignment*> [<*value*>]

Returns

No return value.

Arguments

interface The name of the top-level interface whose assignment is being set.

assignment The assignment whose value is being set.

value (optional) The new assignment value.

Example

```
set_interface_assignment s1 embeddedsw.configuration.isFlash 1
```

Notes

Assignments for Nios II Software Build Tools

Interface assignments provide extra data for the Nios II Software Build Tools working with the generated system.

Assignments for Qsys Pro Tools

There are several assignments that guide behavior in the Qsys Pro tools.

qsys.ui.export_name:	If present, this interface should always be exported when an instance is added to a Qsys Pro system. The value is the requested name of the exported interface in the parent system.
qsys.ui.connect:	If present, this interface should be auto-connected when an instance is added to a Qsys Pro system. The value is a comma-separated list of other interfaces on the same instance that should be connected with this interface.
ui.blockdiagram.direction:	If present, the direction of this interface in the block diagram is set by the user. The value is either "output" or "input".

Related Links

- add_interface on page 764
- get_interface_assignment on page 769
- get_interface_assignments on page 770

13.1.1.12 set_interface_property

Description

Sets the value of a property on an exported top-level interface. You can use this command to set the EXPORT_OF property to specify which interface of a child instance is exported via this top-level interface.

Availability

Main Program, Elaboration, Composition

Usage

`set_interface_property` <*interface*> <*property*> <*value*>

Returns

No return value.

Arguments

interface The name of an exported top-level interface.

property The name of the property Refer to *Interface Properties*.

value The new property value.

Example

```
set_interface_property clk_out EXPORT_OF clk.clk_out
set_interface_property mm_slave linewrapBursts false
```

Notes

The properties for each interface type are different. Refer to the *Avalon Interface Specifications* for more information about interface properties.

Related Links

- get_interface_properties on page 772
- get_interface_property on page 773
- Interface Properties on page 851
- Avalon Interface Specifications

13.1.1.13 set_port_property

Description

Sets a port property.

Availability

Main Program, Elaboration

Usage

`set_port_property` <*port*> <*property*> [<*value*>]

Returns

The new value.

Arguments

port The name of the port.

property One of the supported properties. Refer to *Port Properties*.

value (optional) The value to set.

Example

```
set_port_property rdata WIDTH 32
```

Related Links

- add_interface_port on page 766
- get_port_properties on page 774
- set_port_property on page 779

13.1.1.14 set_interface_upgrade_map

Description

Maps the interface name of an older version of an IP core to the interface name of the current IP core. The interface type must be the same between the older and newer versions of the IP cores. This allows system connections and properties to maintain proper functionality. By default, if the older and newer versions of IP core have the same name and type, then Qsys Pro maintains all properties and connections automatically.

Availability

Parameter Upgrade

Usage

```
set_interface_upgrade_map { <old_interface_name> <new_interface_name>
<old_interface_name_2> <new_interface_name_2> … }
```

Returns

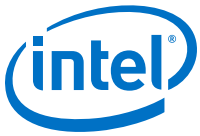
No return value.

Arguments

{ <old_interface_name> <new_interface_name>}	List of mappings between between names of older and newer interfaces.

Example

```
set_interface_upgrade_map { avalon_master_interface new_avalon_master_interface }
```



13.1.2 Parameters

13.1.2.1 add_parameter

Description

Adds a parameter to your IP component.

Availability

Main Program

Usage

`add_parameter <name> <type> [<default_value> <description>]`

Returns

Arguments

name The name of the parameter.

type The data type of the parameter Refer to *Parameter Type Properties*.

default_value (optional) The initial value of the parameter in a new instance of the IP component.

description (optional) Explains the use of the parameter.

Example

```
add_parameter seed INTEGER 17 "The seed to use for data generation."
```

Notes

Most parameter types have a single GUI element for editing the parameter value. `string_list` and `integer_list` parameters are different, because they are edited as tables. A multi-column table can be created by grouping multiple into a single table. To edit multiple list parameters in a single table, the display items for the parameters must be added to a group with a TABLE hint:

```
add_parameter coefficients INTEGER_LIST add_parameter positions
INTEGER_LIST add_display_item "" "Table Group" GROUP TABLE
add_display_item "Table Group" coefficients PARAMETER
add_display_item "Table Group" positions PARAMETER
```

Related Links

- get_parameter_properties on page 784
- get_parameter_property on page 785
- get_parameter_value on page 786
- set_parameter_property on page 790
- set_parameter_value on page 791
- Parameter Type Properties on page 855

13.1.2.2 get_parameters

Description

Returns the names of all the parameters in the IP component.

Availability

Discovery, Main Program, Edit, Elaboration, Validation, Generation, Composition, Fileset Generation, Parameter Upgrade

Usage

```
get_parameters
```

Returns

A list of parameter names

Arguments

No arguments.

Example

```
get_parameters
```

Related Links

- add_parameter on page 782
- get_parameter_property on page 785
- get_parameter_value on page 786
- get_parameters on page 783
- set_parameter_property on page 790

13.1.2.3 get_parameter_properties

Description

Returns a list of all the parameter properties as a list of strings. The `get_parameter_property` and `set_parameter_property` commands are used to get and set the values of these properties, respectively.

Availability

Discovery, Main Program, Edit, Elaboration, Validation, Generation, Composition, Fileset Generation, Parameter Upgrade

Usage

```
get_parameter_properties
```

Returns

A list of parameter property names. Refer to *Parameter Properties*.

Arguments

No arguments.

Example

```
set property_summary [ get_parameter_properties ]
```

Related Links

- add_parameter on page 782
- get_parameter_property on page 785
- get_parameter_value on page 786
- get_parameters on page 783
- set_parameter_property on page 790
- Parameter Properties on page 853

13.1.2.4 get_parameter_property

Description

Returns the value of a property of a parameter.

Availability

Discovery, Main Program, Edit, Elaboration, Validation, Generation, Composition, Fileset Generation, Parameter Upgrade

Usage

`get_parameter_property` <*parameter*> <*property*>

Returns

The value of the property.

Arguments

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

property The name of the property. Refer to *Parameter Properties*.

Example

```
set enabled [ get_parameter_property parameter1 ENABLED ]
```

Related Links

- add_parameter on page 782
- get_parameter_properties on page 784
- get_parameter_value on page 786
- get_parameters on page 783
- set_parameter_property on page 790
- set_parameter_value on page 791
- Parameter Properties on page 853

13.1.2.5 get_parameter_value

Description

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

Availability

Discovery, Edit, Elaboration, Validation, Generation, Composition, Fileset Generation, Parameter Upgrade

Usage

`get_parameter_value` <*parameter*>

Returns

The value of the parameter.

Arguments

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

Example

```
set width [ get_parameter_value fifo_width ]
```

Notes

If `AFFECTS_ELABORATION` is `false` for a given parameter, `get_parameter_value` is not available for that parameter from the elaboration callback. If `AFFECTS_GENERATION` is `false` then it is not available from the generation callback.

Related Links

- add_parameter on page 782
- get_parameter_property on page 785
- get_parameters on page 783
- set_parameter_property on page 790
- set_parameter_value on page 791

13.1.2.6 get_string

Description

Returns the value of an externalized string previously loaded by the `load_strings` command.

Availability

Discovery, Main Program, Edit, Elaboration, Validation, Generation, Composition, Fileset Generation, Parameter Upgrade

Usage

`get_string` <*identifier*>

Returns

The externalized string.

Arguments

identifier The string identifer.

Example

```
hw.tcl:
load_strings test.properties
set_module_property NAME test
set_module_property VERSION [get_string VERSION]
set_module_property DISPLAY_NAME [get_string DISPLAY_NAME]
add_parameter firepower INTEGER 0 ""
set_parameter_property firepower DISPLAY_NAME [get_string PARAM_DISPLAY_NAME]
set_parameter_property firepower TYPE INTEGER
set_parameter_property firepower DESCRIPTION [get_string PARAM_DESCRIPTION]

test.properties:
DISPLAY_NAME = Trogdor!
VERSION = 1.0
PARAM_DISPLAY_NAME = Firepower
PARAM_DESCRIPTION = The amount of force to use when breathing fire.
```

Notes

Use uppercase words separated with underscores to name string identifiers. If you are externalizing module properties, use the module property name for the string identifier:

```
set_module_property DISPLAY_NAME [get_string DISPLAY_NAME]
```

If you are externalizing a parameter property, qualify the parameter property with the parameter name, with uppercase format, if needed:

```
set_parameter_property my_param DISPLAY_NAME [get_string MY_PARAM_DISPLAY_NAME]
```

If you use a string to describe a string format, end the identifier with _FORMAT.

```
set formatted_string [ format  [ get_string TWO_ARGUMENT_MESSAGE_FORMAT ] "arg1"
"arg2" ]
```

Related Links

load_strings on page 789

13.1.2.7 load_strings

Description

Loads strings from an external `.properties` file.

Availability

Discovery, Main Program

Usage

`load_strings` <*path*>

Returns

No return value.

Arguments

path The path to the properties file.

Example

```
hw.tcl:
load_strings test.properties
set_module_property NAME test
set_module_property VERSION [get_string VERSION]
set_module_property DISPLAY_NAME [get_string DISPLAY_NAME]
add_parameter firepower INTEGER 0 ""
set_parameter_property firepower DISPLAY_NAME [get_string PARAM_DISPLAY_NAME]
set_parameter_property firepower TYPE INTEGER
set_parameter_property firepower DESCRIPTION [get_string PARAM_DESCRIPTION]

test.properties:
DISPLAY_NAME = Trogdor!
VERSION = 1.0
PARAM_DISPLAY_NAME = Firepower
PARAM_DESCRIPTION = The amount of force to use when breathing fire.
```

Notes

Refer to the *Java Properties File* for properties file format. A `.properties` file is a text file with *KEY=value* pairs. For externalized strings, the *KEY* is a string identifier and the *value* is the externalized string.
For example:

```
TROGDOR = A dragon with a big beefy arm
```

Related Links

- get_string on page 787
- Java Properties File

13.1.2.8 set_parameter_property

Description

Sets a single parameter property.

Availability

Main Program, Edit, Elaboration, Validation, Composition

Usage

`set_parameter_property` <*parameter*> <*property*> <*value*>

Returns

Arguments

parameter The name of the parameter that is being set.

property The name of the property. Refer to *Parameter Properties*.

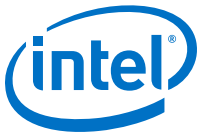
value The new value for the property.

Example

```
set_parameter_property BAUD_RATE ALLOWED_RANGES {9600 19200 38400}
```

Related Links

- add_parameter on page 782
- get_parameter_properties on page 784
- set_parameter_property on page 790
- Parameter Properties on page 853



13.1.2.9 set_parameter_value

Description

Sets a parameter value. The value of a derived parameter can be updated by the IP component in the elaboration callback or the edit callback. Any changes to the value of a derived parameter in the edit callback will not be preserved.

Availability

Edit, Elaboration, Validation, Composition, Parameter Upgrade

Usage

`set_parameter_value` <*parameter*> <*value*>

Returns

No return value.

Arguments

parameter The name of the parameter that is being set.

value Specifies the new parameter value.

Example

```
set_parameter_value half_clock_rate [ expr { [ get_parameter_value clock_rate ] /
2 } ]
```

13.1.2.10 decode_address_map

Description

Converts an XML–formatted address map into a list of Tcl lists. Each inner list is in the correct format for conversion to an array. The XML code that describes each slave includes: its name, start address, and end address.

Availability

Elaboration, Generation, Composition

Usage

`decode_address_map` <*address_map_XML_string*>

Returns

No return value.

Arguments

address_mapXML_string An XML string that describes the address map of a master.

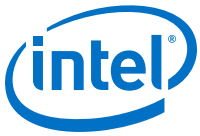
Example

In this example, the code describes the address map for the master that accesses the `ext_ssram`, `sys_clk_timer` and `sysid` slaves. The format of the string may differ from the example below; it may have different white space between the elements and include additional attributes or elements. Use the `decode_address_map` command to decode the code that represents a master's address map to ensure that your code works with future versions of the address map.

```
<address-map>
  <slave name='ext_ssram' start='0x01000000' end='0x01200000' />
  <slave name='sys_clk_timer' start='0x02120800' end='0x02120820' />
  <slave name='sysid' start='0x021208B8' end='0x021208C0' />
</address-map>
```

Note: Intel recommends that you use the code provided below to enumerate over the IP components within an address map, rather than writing your own parser.

```
set address_map_xml [get_parameter_value my_map_param]
set address_map_dec [decode_address_map $address_map_xml]
foreach i $address_map_dec {
    array set info $i
    send_message info "Connected to slave $info(name)"
}
```



13.1.3 Display Items

13.1.3.1 add_display_item

Description

Specifies the following aspects of the IP component display:

- Creates logical groups for an IP component's parameters. For example, to create separate groups for the IP component's timing, size, and simulation parameters. An IP component displays the groups and parameters in the order that you specify the display items in the `_hw.tcl` file.
- Groups a list of parameters to create multi-column tables.
- Specifies an image to provide representation of a parameter or parameter group.
- Creates a button by adding a display item of type `action`. The display item includes the name of the callback to run.

Availability

Main Program

Usage

`add_display_item` <*parent_group*> <*id*> <*type*> [<*args*>]

Returns

Arguments

parent_group Specifies the group to which a display item belongs

id The identifier for the display item. If the item being added is a parameter, this is the parameter name. If the item is a group, this is the group name.

type The type of the display item. Refer to *Display Item Kind Properties*.

args (optional) Provides extra information required for display items.

Example

```
add_display_item "Timing" read_latency PARAMETER
add_display_item "Sounds" speaker_image_id ICON speaker.jpg
```

Notes

The following examples illustrate further illustrate the use of arguments:

- `add_display_item groupName id icon` *`path-to-image-file`*
- `add_display_item groupName parameterName parameter`
- `add_display_item groupName id text "your-text"`

  The your-text argument is a block of text that is displayed in the GUI. Some simple HTML formatting is allowed, such as <b> and <i>, if the text starts with `<html>`.
- `add_display_item parentGroupName childGroupName group [tab]`

  The tab is an optional parameter. If present, the group appears in separate tab in the GUI for the instance.
- `add_display_item parentGroupName actionName action buttonClickCallbackProc`

Related Links

- get_display_item_properties on page 797
- get_display_item_property on page 798
- get_display_items on page 796
- set_display_item_property on page 799
- Display Item Kind Properties on page 860

13.1.3.2 get_display_items

Description

Returns a list of all items to be displayed as part of the parameterization GUI.

Availability

Discovery, Main Program, Edit, Elaboration, Validation, Generation, Composition, Fileset Generation, Parameter Upgrade

Usage

```
get_display_items
```

Returns

List of display item IDs.

Arguments

No arguments.

Example

```
get_display_items
```

Related Links

- add_display_item on page 794
- get_display_item_properties on page 797
- get_display_item_property on page 798
- set_display_item_property on page 799

13.1.3.3 get_display_item_properties

Description

Returns a list of names of the properties of display items that are part of the parameterization GUI.

Availability

Main Program

Usage

```
get_display_item_properties
```

Returns

A list of display item property names. Refer to *Display Item Properties*.

Arguments

No arguments.

Example

```
get_display_item_properties
```

Related Links

- add_display_item on page 794
- get_display_item_property on page 798
- set_display_item_property on page 799
- Display Item Properties on page 859

13.1.3.4 get_display_item_property

Description

Returns the value of a specific property of a display item that is part of the parameterization GUI.

Availability

Main Program, Elaboration, Validation, Composition

Usage

`get_display_item_property` <*display_item*> <*property*>

Returns

The value of a display item property.

Arguments

display_item The id of the display item.

property The name of the property. Refer to *Display Item Properties*.

Example

```
set my_label [get_display_item_property my_action DISPLAY_NAME]
```

Related Links

- add_display_item on page 794
- get_display_item_properties on page 797
- get_display_items on page 796
- set_display_item_property on page 799
- Display Item Properties on page 859

13.1.3.5 set_display_item_property

Description

Sets the value of specific property of a display item that is part of the parameterization GUI.

Availability

Discovery, Main Program, Edit, Elaboration, Validation, Composition

Usage

`set_display_item_property` <*display_item*> <*property*> <*value*>

Returns

No return value.

Arguments

display_item The name of the display item whose property value is being set.

property The property that is being set. Refer to *Display Item Properties*.

value The value to set.

Example

```
set_display_item_property my_action DISPLAY_NAME "Click Me"
set_display_item_property my_action DESCRIPTION "clicking this button runs the
click_me_callback proc in the hw.tcl file"
```

Related Links

- add_display_item on page 794
- get_display_item_properties on page 797
- get_display_item_property on page 798
- Display Item Properties on page 859

13.1.4 Module Definition

13.1.4.1 add_documentation_link

Description

Allows you to link to documentation for your IP component.

Availability

Discovery, Main Program

Usage

`add_documentation_link` <*title*> <*path*>

Returns

No return value.

Arguments

title The title of the document for use on menus and buttons.

path A path to the IP component documentation, using a syntax that provides the entire URL, not a relative path. For example: `http://www.mydomain.com/my_memory_controller.html` or `file:///datasheet.txt`

Example

```
add_documentation_link "Avalon Verification IP Suite User Guide" http://
www.altera.com/literature/ug/ug_avalon_verification_ip.pdf
```

13.1.4.2 get_module_assignment

Description

This command returns the value of an assignment. You can use the `get_module_assignment` and `set_module_assignment` and the `get_interface_assignment` and `set_interface_assignment` commands to provide information about the IP component to embedded software tools and applications.

Availability

Main Program, Elaboration, Validation, Composition

Usage

`get_module_assignment` <*assignment*>

Returns

The value of the assignment

Arguments

assignment The name of the assignment whose value is being retrieved

Example

```
get_module_assignment embeddedsw.CMacro.colorSpace
```

Related Links

- get_module_assignments on page 803
- set_module_assignment on page 808

13.1.4.3 get_module_assignments

Description

Returns the names of the module assignments.

Availability

Main Program, Elaboration, Validation, Composition

Usage

```
get_module_assignments
```

Returns

A list of assignment names.

Arguments

No arguments.

Example

```
get_module_assignments
```

Related Links

- get_module_assignment on page 802
- set_module_assignment on page 808

13.1.4.4 get_module_ports

Description

Returns a list of the names of all the ports which are currently defined.

Availability

Discovery, Main Program, Edit, Elaboration, Validation, Generation, Composition, Fileset Generation, Parameter Upgrade

Usage

```
get_module_ports
```

Returns

A list of port names.

Arguments

No arguments.

Example

```
get_module_ports
```

Related Links

- add_interface on page 764
- add_interface_port on page 766

13.1.4.5 get_module_properties

Description

Returns the names of all the module properties as a list of strings. You can use the `get_module_property` and `set_module_property` commands to get and set values of individual properties. The value returned by this command is always the same for a particular version of Qsys Pro

Availability

Discovery, Main Program, Edit, Elaboration, Validation, Generation, Composition, Fileset Generation, Parameter Upgrade

Usage

```
get_module_properties
```

Returns

List of strings. Refer to *Module Properties*.

Arguments

No arguments.

Example

```
get_module_properties
```

Related Links

- get_module_property on page 806
- set_module_property on page 809
- Module Properties on page 862

13.1.4.6 get_module_property

Description

Returns the value of a single module property.

Availability

Discovery, Main Program, Edit, Elaboration, Validation, Generation, Composition, Fileset Generation, Parameter Upgrade

Usage

`get_module_property` <*property*>

Returns

Various.

Arguments

property The name of the property, Refer to *Module Properties*.

Example

```
set my_name [ get_module_property NAME ]
```

Related Links

- get_module_properties on page 805
- set_module_property on page 809
- Module Properties on page 862

13.1.4.7 send_message

Description

Sends a message to the user of the IP component. The message text is normally interpreted as HTML. You can use the <b> element to provide emphasis. If you do not want the message text to be interpreted as HTML, then pass a list as the message level, for example, `{ Info Text }`.

Availability

Discovery, Main Program, Edit, Elaboration, Validation, Generation, Composition, Fileset Generation, Parameter Upgrade

Usage

`send_message` <*level*> <*message*>

Returns

No return value .

Arguments

level The following message levels are supported:

- **ERROR**--Provides an error message. The Qsys Pro system cannot be generated with existing error messages.
- **WARNING**--Provides a warning message.
- **INFO**--Provides an informational message.
- **PROGRESS**--Reports progress during generation.
- **DEBUG**--Provides a debug message when debug mode is enabled.

message The text of the message.

Example

```
send_message ERROR "The system is down!"
send_message { Info Text } "The system is up!"
```

13.1.4.8 set_module_assignment

Description

Sets the value of the specified assignment.

Availability

Main Program, Elaboration, Validation, Composition

Usage

`set_module_assignment` <*assignment*> [<*value*>]

Returns

No return value.

Arguments

assignment The assignment whose value is being set

value (optional) The value of the assignment

Example

```
set_module_assignment embeddedsw.CMacro.colorSpace CMYK
```

Related Links

- get_module_assignment on page 802
- get_module_assignments on page 803

13.1.4.9 set_module_property

Description

Allows you to set the values for module properties.

Availability

Discovery, Main Program

Usage

`set_module_property` <*property*> <*value*>

Returns

No return value.

Arguments

property The name of the property. Refer to *Module Properties*.

value The new value of the property.

Example

```
set_module_property VERSION 10.0
```

Related Links

- get_module_properties on page 805
- get_module_property on page 806
- Module Properties on page 862

13.1.4.10 add_hdl_instance

Description

Adds an instance of a predefined module, referred to as a *child* or *child instance*. The HDL entity generated from this instance can be instantiated and connected within this IP component's HDL.

Availability

Main Program, Elaboration, Composition

Usage

`add_hdl_instance` <*entity_name*> <*ip_core_type*> [<*version*>]

Returns

The entity name of the added instance.

Arguments

entity_name Specifies a unique local name that you can use to manipulate the instance. This name is used in the generated HDL to identify the instance.

ip_core_type The type refers to a kind of instance available in the IP Catalog, for example `altera_avalon_uart`.

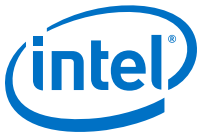
version (optional) The required version of the specified instance type. If no version is specified, the latest version is used.

Example

```
add_hdl_instance my_uart altera_avalon_uart
```

Related Links

- get_instance_parameter_value on page 828
- get_instance_parameters on page 826
- get_instances on page 818
- set_instance_parameter_value on page 831



13.1.4.11 package

Description

Allows you to specify a particular version of the Qsys Pro software to avoid software compatibility issues, and to determine which version of the **_hw.tcl** API to use for the IP component. You must use the package command at the beginning of your **_hw.tcl** file.

Availability

Main Program

Usage

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

Returns

No return value

Arguments

version The version of Qsys Pro that you require, such as 14.1.

Example

```
package require -exact qsys 14.1
```

13.1.5 Composition

13.1.5.1 add_instance

Description

Adds an instance of an IP component, referred to as a child or child instance to the subsystem. You can use this command to create IP components that are composed of other IP component instances. The HDL for this subsystem will be generated; no custom HDL will need to be written for the IP component.

Availability

Main Program, Composition

Usage

`add_instance` <*name*> <*type*> [<*version*>]

Returns

No return value.

Arguments

name Specifies a unique local name that you can use to manipulate the instance. This name is used in the generated HDL to identify the instance.

type The type refers to a type available in the IP Catalog, for example `altera_avalon_uart`.

version (optional) The required version of the specified type. If no version is specified, the highest available version is used.

Example

```
add_instance my_uart altera_avalon_uart
add_instance my_uart altera_avalon_uart 14.1
```

Related Links

- add_connection on page 814
- get_instance_interface_property on page 825
- get_instance_parameter_value on page 828
- get_instance_parameters on page 826
- get_instance_property on page 822
- get_instances on page 818
- set_instance_parameter_value on page 831

13.1.5.2 add_connection

Description

Connects the named interfaces on child instances together using an appropriate connection type. Both interface names consist of a child instance name, followed by the name of an interface provided by that module. For example, `mux0.out` is the interface named `out` on the instance named `mux0`. Be careful to connect the start to the end, and not the other way around.

Availability

Main Program, Composition

Usage

`add_connection` <*start*> [<*end*> <*kind*> <*name*>]

Returns

The name of the newly added connection in `start.point/end.point` format.

Arguments

start The start interface to be connected, in `<instance_name>.<interface_name>` format.

end (optional) The end interface to be connected, `<instance_name>.<interface_name>`.

kind (optional) The type of connection, such as `avalon` or `clock`.

name (optional) A custom name for the connection. If unspecified, the name will be `<start_instance>.<interface>.<end_instance><interface>`

Example

```
add_connection dma.read_master sdram.s1 avalon
```

Related Links

- add_instance on page 813
- get_instance_interfaces on page 819

13.1.5.3 get_connections

Description

Returns a list of all connections in the composed subsystem.

Availability

Main Program, Composition

Usage

```
get_connections
```

Returns

A list of connections.

Arguments

No arguments.

Example

```
set all_connections [ get_connections ]
```

Related Links

add_connection on page 814

13.1.5.4 get_connection_parameters

Description

Returns a list of parameters found on a connection.

Availability

Main Program, Composition

Usage

`get_connection_parameters` <*connection*>

Returns

A list of parameter names

Arguments

connection The connection to query.

Example

```
get_connection_parameters cpu.data_master/dma0.csr
```

Related Links

- add_connection on page 814
- get_connection_parameter_value on page 817

13.1.5.5 get_connection_parameter_value

Description

Returns the value of a parameter on the connection. Parameters represent aspects of the connection that can be modified once the connection is created, such as the base address for an Avalon Memory Mapped connection.

Availability

Composition

Usage

`get_connection_parameter_value` <*connection*> <*parameter*>

Returns

The value of the parameter.

Arguments

connection The connection to query.

parameter The name of the parameter.

Example

```
get_connection_parameter_value cpu.data_master/dma0.csr baseAddress
```

Related Links

- add_connection on page 814
- get_connection_parameters on page 816

13.1.5.6 get_instances

Description

Returns a list of the instance names for all child instances in the system.

Availability

Main Program, Elaboration, Validation, Composition

Usage

```
get_instances
```

Returns

A list of child instance names.

Arguments

No arguments.

Example

```
get_instances
```

Notes

This command can be used with instances created by either `add_instance` or `add_hdl_instance`.

Related Links

- add_hdl_instance on page 810
- add_instance on page 813
- get_instance_parameter_value on page 828
- get_instance_parameters on page 826
- set_instance_parameter_value on page 831

13.1.5.7 get_instance_interfaces

Description

Returns a list of interfaces found in a child instance. The list of interfaces can change if the parameterization of the instance changes.

Availability

Validation, Composition

Usage

`get_instance_interfaces` <*instance*>

Returns

A list of interface names.

Arguments

instance The name of the child instance.

Example

```
get_instance_interfaces pixel_converter
```

Related Links

- add_instance on page 813
- get_instance_interface_ports on page 820
- get_instance_interfaces on page 819

13.1.5.8 get_instance_interface_ports

Description

Returns a list of ports found in an interface of a child instance.

Availability

Validation, Composition, Fileset Generation

Usage

`get_instance_interface_ports` <*instance*> <*interface*>

Returns

A list of port names found in the interface.

Arguments

instance The name of the child instance.

interface The name of an interface on the child instance.

Example

```
set port_names [ get_instance_interface_ports cpu data_master ]
```

Related Links

- add_instance on page 813
- get_instance_interfaces on page 819
- get_instance_port_property on page 829

13.1.5.9 get_instance_interface_properties

Description

Returns the names of all of the properties of the specified interface

Availability

Validation, Composition

Usage

`get_instance_interface_properties` <*instance*> <*interface*>

Returns

List of property names.

Arguments

instance The name of the child instance.

interface The name of an interface on the instance.

Example

```
set properties [ get_instance_interface_properties cpu data_master ]
```

Related Links

- add_instance on page 813
- get_instance_interface_property on page 825
- get_instance_interfaces on page 819

13.1.5.10 get_instance_property

Description

Returns the value of a single instance property.

Availability

Main Program, Elaboration, Validation, Composition, Fileset Generation

Usage

`get_instance_property` <*instance*> <*property*>

Returns

Various.

Arguments

instance The name of the instance.

property The name of the property. Refer to *Instance Properties*.

Example

```
set my_name [ get_instance_property myinstance NAME ]
```

Related Links

- add_instance on page 813
- get_instance_properties on page 824
- set_instance_property on page 823
- Instance Properties on page 852

13.1.5.11 set_instance_property

Description

Allows a user to set the properties of a child instance.

Availability

Main Program, Elaboration, Validation, Composition

Usage

`set_instance_property` <*instance*> <*property*> <*value*>

Returns

Arguments

instance The name of the instance.

property The name of the property to set. Refer to *Instance Properties*.

value The new property value.

Example

```
set_instance_property myinstance SUPRESS_ALL_WARNINGS true
```

Related Links

- add_instance on page 813
- get_instance_properties on page 824
- get_instance_property on page 822
- Instance Properties on page 852

13.1.5.12 get_instance_properties

Description

Returns the names of all the instance properties as a list of strings. You can use the `get_instance_property` and `set_instance_property` commands to get and set values of individual properties. The value returned by this command is always the same for a particular version of Qsys Pro

Availability

Discovery, Main Program, Edit, Elaboration, Validation, Generation, Composition, Fileset Generation, Parameter Upgrade

Usage

```
get_instance_properties
```

Returns

List of strings. Refer to *Instance Properties*.

Arguments

No arguments.

Example

```
get_instance_properties
```

Related Links

- add_instance on page 813
- get_instance_property on page 822
- set_instance_property on page 823
- Instance Properties on page 852

13.1.5.13 get_instance_interface_property

Description

Returns the value of a property for an interface in a child instance.

Availability

Validation, Composition

Usage

`get_instance_interface_property` <*instance*> <*interface*> <*property*>

Returns

The value of the property.

Arguments

instance The name of the child instance.

interface The name of an interface on the child instance.

property The name of the property of the interface.

Example

```
set value [ get_instance_interface_property cpu data_master setupTime ]
```

Related Links

- add_instance on page 813
- get_instance_interfaces on page 819

13.1.5.14 get_instance_parameters

Description

Returns a list of names of the parameters on a child instance that can be set using `set_instance_parameter_value`. It omits parameters that are derived and those that have the `SYSTEM_INFO` parameter property set.

Availability

Main Program, Elaboration, Validation, Composition

Usage

`get_instance_parameters` <*instance*>

Returns

A list of parameters in the instance.

Arguments

instance The name of the child instance.

Example

```
set parameters [ get_instance_parameters instance ]
```

Notes

You can use this command with instances created by either `add_instance` or `add_hdl_instance`.

Related Links

- add_hdl_instance on page 810
- add_instance on page 813
- get_instance_parameter_value on page 828
- get_instances on page 818
- set_instance_parameter_value on page 831

13.1.5.15 get_instance_parameter_property

Description

Returns the value of a property on a parameter in a child instance. Parameter properties are metadata about how the parameter will be used by the Qsys Pro tools.

Availability

Validation, Composition

Usage

`get_instance_parameter_property` <*instance*> <*parameter*> <*property*>

Returns

The value of the parameter property.

Arguments

instance The name of the child instance.

parameter The name of the parameter in the instance.

property The name of the property of the parameter. Refer to *Parameter Properties*.

Example

```
get_instance_parameter_property instance parameter property
```

Related Links

- add_instance on page 813
- Parameter Properties on page 853

13.1.5.16 get_instance_parameter_value

Description

Returns the value of a parameter in a child instance. You cannot use this command to get the value of parameters whose values are derived or those that are defined using the SYSTEM_INFO parameter property.

Availability

Elaboration, Validation, Composition

Usage

`get_instance_parameter_value` <*instance*> <*parameter*>

Returns

The value of the parameter.

Arguments

instance The name of the child instance.

parameter Specifies the parameter whose value is being retrieved.

Example

```
set dpi [ get_instance_parameter_value pixel_converter input_DPI ]
```

Notes

You can use this command with instances created by either `add_instance` or `add_hdl_instance`.

Related Links

- add_hdl_instance on page 810
- add_instance on page 813
- get_instance_parameters on page 826
- get_instances on page 818
- set_instance_parameter_value on page 831

13.1.5.17 get_instance_port_property

Description

Returns the value of a property of a port contained by an interface in a child instance.

Availability

Validation, Composition, Fileset Generation

Usage

`get_instance_port_property` <*instance*> <*port*> <*property*>

Returns

The value of the property for the port.

Arguments

instance The name of the child instance.

port The name of a port in one of the interfaces on the child instance.

property The property whose value is being retrieved. Only the following port properties can be queried on ports of child instances: `ROLE`, `DIRECTION`, `WIDTH`, `WIDTH_EXPR` and `VHDL_TYPE`. Refer to *Port Properties*.

Example

```
get_instance_port_property instance port property
```

Related Links

- add_instance on page 813
- get_instance_interface_ports on page 820
- Port Properties on page 857

13.1.5.18 set_connection_parameter_value

Description

Sets the value of a parameter of the connection. The start and end are each interface names of the format `<instance>.<interface>`. Connection parameters depend on the type of connection, for Avalon-MM they include base addresses and arbitration priorities.

Availability

Main Program, Composition

Usage

`set_connection_parameter_value` <*connection*> <*parameter*> <*value*>

Returns

No return value.

Arguments

connection Specifies the name of the connection as returned by the `add_conection`command. It is of the form `start.point/end.point`.

parameter The name of the parameter.

value The new parameter value.

Example

```
set_connection_parameter_value cpu.data_master/dma0.csr baseAddress "0x000a0000"
```

Related Links

- add_connection on page 814
- get_connection_parameter_value on page 817

13.1.5.19 set_instance_parameter_value

Description

Sets the value of a parameter for a child instance. Derived parameters and SYSTEM_INFO parameters for the child instance can not be set with this command.

Availability

Main Program, Elaboration, Composition

Usage

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

Returns

Vo return value.

Arguments

instance Specifies the name of the child instance.

parameter Specifies the parameter that is being set.

value Specifies the new parameter value.

Example

```
set_instance_parameter_value uart_0 baudRate 9600
```

Notes

You can use this command with instances created by either `add_instance` or `add_hdl_instance`.

Related Links

- add_hdl_instance on page 810
- add_instance on page 813
- get_instance_parameter_value on page 828
- get_instances on page 818

13.1.6 Fileset Generation

13.1.6.1 add_fileset

Description

Adds a generation fileset for a particular target as specified by the `kind`. Qsys Pro calls the target (`SIM_VHDL`, `SIM_VERILOG`, `QUARTUS_SYNTH`, or `EXAMPLE_DESIGN`) when the specified generation target is requested. You can define multiple filesets for each kind of fileset. Qsys Pro passes a single argument to the specified callback procedure. The value of the argument is a generated name, which you must use in the top-level module or entity declaration of your IP component. To override this generated name, you can set the fileset property `TOP_LEVEL`.

Availability

Main Program

Usage

`add_fileset` <*name*> <*kind*> [<*callback_proc*> <*display_name*>]

Returns

No return value.

Arguments

name The name of the fileset.

kind The kind of fileset. Refer to *Fileset Properties*.

callback_proc (optional) A string identifying the name of the callback procedure. If you add files in the global section, you can then specify a blank callback procedure.

display_name (optional) A display string to identify the fileset.

Example

```
add_fileset my_synthesis_fileset QUARTUS_SYNTH mySynthCallbackProc "My Synthesis"
proc mySynthCallbackProc { topLevelName } { ... }
```

Notes

If using the `TOP_LEVEL` fileset property, all parameterizations of the component must use identical HDL.

Related Links

- add_fileset_file on page 834
- get_fileset_property on page 839
- Fileset Properties on page 864

13.1.6.2 add_fileset_file

Description

Adds a file to the generation directory. You can specify source file locations with either an absolute path, or a path relative to the IP component's `_hw.tcl` file. When you use the `add_fileset_file` command in a fileset callback, the Quartus Prime software compiles the files in the order that they are added.

Availability

Main Program, Fileset Generation

Usage

`add_fileset_file` <*output_file*> <*file_type*> <*file_source*> <*path_or_contents*> [<*attributes*>]

Returns

No return value.

Arguments

output_file Specifies the location to store the file after Qsys Pro generation

file_type The kind of file. Refer to *File Kind Properties*.

file_source Specifies whether the file is being added by path, or by file contents. Refer to *File Source Properties*.

path_or_contents When the `file_source` is PATH, specifies the file to be copied to `output_file`. When the `file_source` is TEXT, specifies the text contents to be stored in the file.

attributes (optional) An optional list of file attributes. Typically used to specify that a file is intended for use only in a particular simulator. Refer to *File Attribute Properties*.

Example

```
add_fileset_file "./implementation/rx_pma.sv" SYSTEM_VERILOG PATH synth_rx_pma.sv
add_fileset_file gui.sv SYSTEM_VERILOG TEXT "Customize your IP core"
```

Related Links

- add_fileset on page 833
- get_fileset_file_attribute on page 836
- File Kind Properties on page 868
- File Source Properties on page 869
- File Attribute Properties on page 867

13.1.6.3 set_fileset_property

Description

Allows you to set the properties of a fileset.

Availability

Main Program, Elaboration, Fileset Generation

Usage

`set_fileset_property` <*fileset*> <*property*> <*value*>

Returns

No return value.

Arguments

fileset The name of the fileset.

property The name of the property to set. Refer to *Fileset Properties*.

value The new property value.

Example

```
set_fileset_property mySynthFileset TOP_LEVEL simple_uart
```

Notes

When a fileset callback is called, the callback procedure will be passed a single argument. The value of this argument is a generated name which must be used in the top-level module or entity declaration of your IP component. If set, the TOP_LEVEL specifies a fixed name for the top-level name of your IP component.

The TOP_LEVEL property must be set in the global section. It cannot be set in a fileset callback.

If using the TOP_LEVEL fileset property, all parameterizations of the IP component must use identical HDL.

Related Links

- add_fileset on page 833
- Fileset Properties on page 864

13.1.6.4 get_fileset_file_attribute

Description

Returns the attribute of a fileset file.

Availability

Main Program, Fileset Generation

Usage

`get_fileset_file_attribute` <*output_file*> <*attribute*>

Returns

Value of the fileset File attribute.

Arguments

output_file Location of the output file.

attribute Specifies the name of the attribute Refer to *File Attribute Properties*.

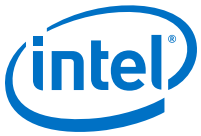
Example

```
get_fileset_file_attribute my_file.sv ALDEC_SPECIFIC
```

Related Links

- add_fileset on page 833
- add_fileset_file on page 834
- get_fileset_file_attribute on page 836
- File Attribute Properties on page 867

- add_fileset on page 833
- add_fileset_file on page 834
- get_fileset_file_attribute on page 836
- File Attribute Properties on page 867



13.1.6.5 set_fileset_file_attribute

Description

Sets the attribute of a fileset file.

Availability

Main Program, Fileset Generation

Usage

`set_fileset_file_attribute` <*output_file*> <*attribute*> <*value*>

Returns

The attribute value if it was set.

Arguments

output_file Location of the output file.

attribute Specifies the name of the attribute Refer to *File Attribute Properties*.

value Value to set the attribute to.

Example

```
set_fileset_file_attribute my_file_pkg.sv COMMON_SYSTEMVERILOG_PACKAGE
my_file_package
```

13.1.6.6 get_fileset_properties

Description

Returns a list of properties that can be set on a fileset.

Availability

Discovery, Main Program, Edit, Elaboration, Validation, Generation, Composition, Fileset Generation, Parameter Upgrade

Usage

```
get_fileset_properties
```

Returns

A list of property names. Refer to *Fileset Properties*.

Arguments

No arguments.

Example

```
get_fileset_properties
```

Related Links

- add_fileset on page 833
- get_fileset_properties on page 838
- set_fileset_property on page 835
- Fileset Properties on page 864

13.1.6.7 get_fileset_property

Description

Returns the value of a fileset property for a fileset.

Availability

Main Program, Elaboration, Fileset Generation

Usage

`get_fileset_property` <*fileset*> <*property*>

Returns

The value of the property.

Arguments

fileset The name of the fileset.

property The name of the property to query. Refer to *Fileset Properties*.

Example

```
get_fileset_property fileset property
```

Related Links

Fileset Properties on page 864

13.1.6.8 get_fileset_sim_properties

Description

Returns simulator properties for a fileset.

Availability

Main Program, Fileset Generation

Usage

`get_fileset_sim_properties` <*fileset*> <*platform*> <*property*>

Returns

The fileset simulator properties.

Arguments

fileset The name of the fileset.

platform The operating system for that applies to the property. Refer to *Operating System Properties*.

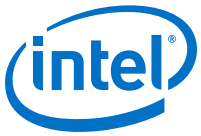
property Specifies the name of the property to set. Refer to *Simulator Properties*.

Example

```
get_fileset_sim_properties my_fileset LINUX64 OPT_CADENCE_64BIT
```

Related Links

- add_fileset on page 833
- set_fileset_sim_properties on page 841
- Operating System Properties on page 876
- Simulator Properties on page 870



13.1.6.9 set_fileset_sim_properties

Description

Sets simulator properties for a given fileset

Availability

Main Program, Fileset Generation

Usage

`set_fileset_sim_properties` <*fileset*> <*platform*> <*property*> <*value*>

Returns

The fileset simulator properties if they were set.

Arguments

fileset The name of the fileset.

platform The operating system that applies to the property. Refer to *Operating System Properties*.

property Specifies the name of the property to set. Refer to *Simulator Properties*.

value Specifies the value of the property.

Example

```
set_fileset_sim_properties my_fileset LINUX64 OPT_MENTOR_PLI "{libA} {libB}"
```

Related Links

- get_fileset_sim_properties on page 840
- Operating System Properties on page 876
- Simulator Properties on page 870

13.1.6.10 create_temp_file

Description

Creates a temporary file, which you can use inside the fileset callbacks of a `_hw.tcl` file. This temporary file is included in the generation output if it is added using the `add_fileset_file` command.

Availability

Fileset Generation

Usage

`create_temp_file` <*path*>

Returns

The path to the temporary file.

Arguments

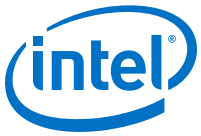
path The name of the temporary file.

Example

```
set filelocation [create_temp_file "./hdl/compute_frequency.v" ]
add_fileset_file compute_frequency.v VERILOG PATH ${filelocation}
```

Related Links

- add_fileset on page 833
- add_fileset_file on page 834



13.1.7 Miscellaneous

13.1.7.1 check_device_family_equivalence

Description

Returns 1 if the device family is equivalent to one of the families in the device families lis., Returns 0 if the device family is not equivalent to any families. This command ignores differences in capitalization and spaces.

Availability

Discovery, Main Program, Edit, Elaboration, Validation, Composition, Fileset Generation, Parameter Upgrade

Usage

`check_device_family_equivalence` <*device_family*> <*device_family_list*>

Returns

1 if equivalent, 0 if not equivalent.

Arguments

device_family The device family name that is being checked.

device_family_list The list of device family names to check against.

Example

```
check_device_family_equivalence "CYLCONE III LS" { "stratixv" "Cyclone IV"
"cycloneiiils" }
```

Related Links

get_device_family_displayname on page 845

13.1.7.2 get_device_family_displayname

Description

Returns the display name of a given device family.

Availability

Discovery, Main Program, Edit, Elaboration, Validation, Composition, Fileset Generation, Parameter Upgrade

Usage

`get_device_family_displayname` <*device_family*>

Returns

The preferred display name for the device family.

Arguments

device_family A device family name.

Example

```
get_device_family_displayname cycloneiiils ( returns: "Cyclone IV LS" )
```

Related Links

check_device_family_equivalence on page 844

13.1.7.3 get_qip_strings

Description

Returns a Tcl list of QIP strings for the IP component.

Availability

Discovery, Main Program, Edit, Elaboration, Validation, Composition, Parameter Upgrade

Usage

```
get_qip_strings
```

Returns

A Tcl list of qip strings set by this IP component.

Arguments

No arguments.

Example

```
set strings [ get_qip_strings ]
```

Related Links

set_qip_strings on page 847

13.1.7.4 set_qip_strings

Description

Places strings in the Quartus Prime IP File (**.qip**) file, which Qsys Pro passes to the command as a Tcl list. You add the **.qip** file to your Quartus Prime project on the **Files** page, in the **Settings** dialog box. Successive calls to `set_qip_strings` are not additive and replace the previously declared value.

Availability

Discovery, Main Program, Edit, Elaboration, Validation, Composition, Parameter Upgrade

Usage

`set_qip_strings` <*qip_strings*>

Returns

The Tcl list which was set.

Arguments

qip_strings A space-delimited Tcl list.

Example

```
set_qip_strings {"QIP Entry 1" "QIP Entry 2"}
```

Notes

You can use the following macros in your QIP strings entry:

%entityName%	The generated name of the entity replaces this macro when the string is written to the **.qip** file.
%libraryName%	The compilation library this IP component was compiled into is inserted in place of this macro inside the **.qip** file.
%instanceName%	The name of the instance is inserted in place of this macro inside the `.qip` file.

Related Links

get_qip_strings on page 846

13.1.7.5 set_interconnect_requirement

Description

Sets the value of an interconnect requirement for a system or an interface on a child instance.

Availability

Composition

Usage

`set_interconnect_requirement` <*element_id*> <*name*> <*value*>

Returns

No return value

Arguments

element_id `{$system}` for system requirements, or qualified name of the interface of an instance, in `<instance>.<interface>` format. Note that the system identifier has to be escaped in TCL.

name The name of the requirement.

value The new requirement value.

Example

```
set_interconnect_requirement {$system} qsys_mm.maxAdditionalLatency 2
```

13.2 Qsys Pro _hw.tcl Property Reference

13.2.1 Script Language Properties

Name	Description
TCL	Implements the script in Tcl.

13.2.2 Interface Properties

Name	Description
CMSIS_SVD_FILE	Specifies the connection point's associated CMSIS file.
CMSIS_SVD_VARIABLES	Defines the variables inside a .svd file.
ENABLED	Specifies whether or not interface is enabled.
EXPORT_OF	For composed _hwl.tcl files, the EXPORT_OF property indicates which interface of a child instance is to be exported through this interface. Before using this command, you must have created the border interface using add_interface. The interface to be exported is of the form <*instanceName.interfaceName*>. Example: set_interface_property CSC_input EXPORT_OF my_colorSpaceConverter.input_port
PORT_NAME_MAP	A map of external port names to internal port names, formatted as a Tcl list. Example: set_interface_property <interface name> PORT_NAME_MAP "<new port name> <old port name> <new port name 2> <old port name 2>"
SVD_ADDRESS_GROUP	Generates a CMSIS SVD file. Masters in the same SVD address group will write register data of their connected slaves into the same SVD file
SVD_ADDRESS_OFFSET	Generates a CMSIS SVD file. Slaves connected to this master will have their base address offset by this amount in the SVD file.

13.2.3 Instance Properties

Name	Description
HDLINSTANCE_GET_GENERATED_NAME	Qsys Pro uses this property to get the auto-generated fixed name when the instance property HDLINSTANCE_USE_GENERATED_NAME is set to true, and only applies to fileSet callbacks.
HDLINSTANCE_USE_GENERATED_NAME	If true, instances added with the add_hdl_instance command are instructed to use unique auto-generated fixed names based on the parameterization.
SUPPRESS_ALL_INFO_MESSAGES	If true, allows you to suppress all Info messages that originate from a child instance.
SUPPRESS_ALL_WARNINGS	If true, allows you to suppress alL warnings that originate from a child instance

13.2.4 Parameter Properties

Type	Name	Description
Boolean	AFFECTS_ELABORATION	Set AFFECTS_ELABORATION to false for parameters that do not affect the external interface of the module. An example of a parameter that does not affect the external interface is isNonVolatileStorage. An example of a parameter that does affect the external interface is width. When the value of a parameter changes, if that parameter has set AFFECTS_ELABORATION=false, the elaboration phase (calling the callback or hardware analysis) is not repeated, improving performance. Because the default value of AFFECTS_ELABORATION is true, the provided HDL file is normally re-analyzed to determine the new port widths and configuration every time a parameter changes.
Boolean	AFFECTS_GENERATION	The default value of AFFECTS_GENERATION is false if you provide a top-level HDL module; it is true if you provide a fileset callback. Set AFFECTS_GENERATION to false if the value of a parameter does not change the results of fileset generation.
Boolean	AFFECTS_VALIDATION	The AFFECTS_VALIDATION property marks whether a parameter's value is used to set derived parameters, and whether the value affects validation messages. When set to false, this may improve response time in the parameter editor UI when the value is changed.
String[]	ALLOWED_RANGES	Indicates the range or ranges that the parameter value can have. For integers, The ALLOWED_RANGES property is a list of ranges that the parameter can take on, where each range is a single value, or a range of values defined by a start and end value separated by a colon, such as 11:15. This property can also specify legal values and display strings for integers, such as {0:None 1:Monophonic 2:Stereo 4:Quadrophonic} meaning 0, 1, 2, and 4 are the legal values. You can also assign display strings to be displayed in the parameter editor for string variables. For example, ALLOWED_RANGES {"dev1:Cyclone IV GX""dev2:Stratix V GT"}.
String	DEFAULT_VALUE	The default value.
Boolean	DERIVED	When true, indicates that the parameter value can only be set by the IP component, and cannot be set by the user. Derived parameters are not saved as part of an instance's parameter values. The default value is false.
String	DESCRIPTION	A short user-visible description of the parameter, suitable for a tooltip description in the parameter editor.
String[]	DISPLAY_HINT	Provides a hint about how to display a property. The following values are possible: • boolean--for integer parameters whose value can be 0 or 1. The parameter displays as an option that you can turn on or off. • radio--displays a parameter with a list of values as radio buttons instead of a drop-down list. • hexadecimal--for integer parameters, display and interpret the value as a hexadecimal number, for example: 0x00000010 instead of 16. • fixed_size--for string_list and integer_list parameters, the fixed_size DISPLAY_HINT eliminates the **add** and **remove** buttons from tables.
String	DISPLAY_NAME	This is the GUI label that appears to the left of this parameter.
String	DISPLAY_UNITS	This is the GUI label that appears to the right of the parameter.
Boolean	ENABLED	When false, the parameter is disabled, meaning that it is displayed, but greyed out, indicating that it is not editable on the parameter editor.

Type	Name	Description
String	GROUP	Controls the layout of parameters in GUI
Boolean	HDL_PARAMETER	When true, the parameter must be passed to the HDL IP component description. The default value is `false`.
String	LONG_DESCRIPTION	A user-visible description of the parameter. Similar to DESCRIPTION, but allows for a more detailed explanation.
String	NEW_INSTANCE_VALUE	This property allows you to change the default value of a parameter without affecting older IP components that have did not explicitly set a parameter value, and use the DEFAULT_VALUE property. The practical result is that older instances will continue to use DEFAULT_VALUE for the parameter and new instances will use the value assigned by NEW_INSTANCE_VALUE.
String[]	SYSTEM_INFO	Allows you to assign information about the instantiating system to a parameter that you define. SYSTEM_INFO requires an argument specifying the type of information requested, *<info-type>*.
String	SYSTEM_INFO_ARG	Defines an argument to be passed to a particular SYSTEM_INFO function, such as the name of a reset interface.
(various)	SYSTEM_INFO_TYPE	Specifies one of the types of system information that can be queried. Refer to *System Info Type Properties*.
(various)	TYPE	Specifies the type of the parameter. Refer to *Parameter Type Properties*.
(various)	UNITS	Sets the units of the parameter. Refer to *Units Properties*.
Boolean	VISIBLE	Indicates whether or not to display the parameter in the parameterization GUI.
String	WIDTH	For a STD_LOGIC_VECTOR parameter, this indicates the width of the logic vector.

Related Links

- System Info Type Properties on page 872
- Parameter Type Properties on page 855
- Units Properties on page 875

13.2.5 Parameter Type Properties

Name	Description
BOOLEAN	A boolean parameter whose value is `true` or `false`.
FLOAT	A signed 32-bit floating point parameter. Not supported for HDL parameters.
INTEGER	A signed 32-bit integer parameter.
INTEGER_LIST	A parameter that contains a list of 32-bit integers. Not supported for HDL parameters.
LONG	A signed 64-bit integer parameter. Not supported for HDL parameters.
NATURAL	A 32-bit number that contain values `0` to `2147483647` (`0x7fffffff`).
POSITIVE	A 32-bit number that contains values `1` to `2147483647` (`0x7fffffff`).
STD_LOGIC	A single bit parameter whose value can be `1` or `0`;
STD_LOGIC_VECTOR	An arbitrary-width number. The parameter property `WIDTH` determines the size of the logic vector.
STRING	A string parameter.
STRING_LIST	A parameter that contains a list of strings. Not supported for HDL parameters.

13.2.6 Parameter Status Properties

Type	Name	Description
Boolean	ACTIVE	Indicates the parameter is a regular parameter.
Boolean	DEPRECATED	Indicates the parameter exists only for backwards compatibility, and may not have any effect.
Boolean	EXPERIMENTAL	Indicates the parameter is experimental, and not exposed in the design flow.

13.2.7 Port Properties

Type	Name	Description
(various)	DIRECTION	The direction of the port from the IP component's perspective. Refer to *Direction Properties*.
String	DRIVEN_BY	Indicates that this output port is always driven to a constant value or by an input port. If all outputs on an IP component specify a driven_by property, the HDL for the IP component will be generated automatically.
String[]	FRAGMENT_LIST	This property can be used in 2 ways: First you can take a single RTL signal and split it into multiple Qsys Pro signals add_interface_port <interface> foo <role> <direction> <width> add_interface_port <interface> bar <role> <direction> <width> set_port_property foo fragment_list "my_rtl_signal(3:0)" set_port_property bar fragment_list "my_rtl_signal(6:4)" Second you can take multiple RTL signals and combine them into a single Qsys Pro signal add_interface_port <interface> baz <role> <direction> <width> set_port_property baz fragment_list "rtl_signal_1(3:0) rtl_signal_2(3:0)" Note: The listed bits in a port fragment must match the declared width of the Qsys Pro signal.
String	ROLE	Specifies an Avalon signal type such as waitrequest, readdata, or read. For a complete list of signal types, refer to the *Avalon Interface Specifications*.
Boolean	TERMINATION	When true, instead of connecting the port to the Qsys Pro system, it is left unconnected for output and bidir or set to a fixed value for input. Has no effect for IP components that implement a generation callback instead of using the default wrapper generation.
BigInteger	TERMINATION_VALUE	The constant value to drive an input port.
(various)	VHDL_TYPE	Indicates the type of a VHDL port. The default value, auto, selects std_logic if the width is fixed at 1, and std_logic_vector otherwise. Refer to *Port VHDL Type Properties*.
String	WIDTH	The width of the port in bits. Cannot be set directly. Any changes must be set through the WIDTH_EXPR property.
String	WIDTH_EXPR	The width expression of a port. The width_value_expr property can be set directly to a numeric value if desired. When get_port_property is used width always returns the current integer width of the port while width_expr always returns the unevaluated width expression.
Integer	WIDTH_VALUE	The width of the port in bits. Cannot be set directly. Any changes must be set through the WIDTH_EXPR property.

Related Links

- Direction Properties on page 858
- Port VHDL Type Properties on page 871
- Avalon Interface Specifications

13.2.8 Direction Properties

Name	Description
`Bidir`	Direction for a bidirectional signal.
`Input`	Direction for an input signal.
`Output`	Direction for an output signal.

13.2.9 Display Item Properties

Type	Name	Description
String	DESCRIPTION	A description of the display item, which you can use as a tooltip.
String[]	DISPLAY_HINT	A hint that affects how the display item displays in the parameter editor.
String	DISPLAY_NAME	The label for the display item in a the parameter editor.
Boolean	ENABLED	Indicates whether the display item is enabled or disabled.
String	PATH	The path to a file. Only applies to display items of type ICON.
String	TEXT	Text associated with a display item. Only applies to display items of type TEXT.
Boolean	VISIBLE	Indicates whether this display item is visible or not.

13.2.10 Display Item Kind Properties

Name	Description
ACTION	An action displays as a button in the GUI. When the button is clicked, it calls the callback procedure. The button label is the display item `id`.
GROUP	A group that is a child of the `parent_group` group. If the `parent_group` is an empty string, this is a top-level group.
ICON	A `.gif`, `.jpg`, or `.png` file.
PARAMETER	A parameter in the instance.
TEXT	A block of text.

13.2.11 Display Hint Properties

Name	Description
BIT_WIDTH	Bit width of a number
BOOLEAN	Integer value either 0 or 1.
COLLAPSED	Indicates whether a group is collapsed when initially displayed.
COLUMNS	Number of columns in text field, for example, "columns:N".
EDITABLE	Indicates whether a list of strings allows free-form text entry (editable combo box).
FILE	Indicates that the string is an optional file path, for example, "file:jpg,png,gif".
FIXED_SIZE	Indicates a fixed size for a table or list.
GROW	if set, the widget can grow when the IP component is resized.
HEXADECIMAL	Indicates that the long integer is hexadecimal.
RADIO	Indicates that the range displays as radio buttons.
ROWS	Number of rows in text field, or visible rows in a table, for example, "rows:N".
SLIDER	Range displays as slider.
TAB	if present for a group, the group displays in a tab
TABLE	if present for a group, the group must contain all list-type parameters, which display collectively in a single table.
TEXT	String is a text field with a limited character set, for example, "text:A-Za-z0-9_".
WIDTH	width of a table column

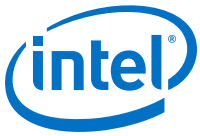
13.2.12 Module Properties

Name	Description
ANALYZE_HDL	When set to false, prevents a call to the Quartus Prime mapper to verify port widths and directions, speeding up generation time at the expense of fewer validation checks. If this property is set to false, invalid port widths and directions are discovered during the Quartus Prime software compilation. This does not affect IP components using filesets to manage synthesis files.
AUTHOR	The IP component author.
COMPOSITION_CALLBACK	The name of the composition callback. If you define a composition callback, you cannot not define the generation or elaboration callbacks.
DATASHEET_URL	Deprecated. Use `add_documentation_link` to provide documentation links.
DESCRIPTION	The description of the IP component, such as "This IP component puts the shizzle in the frobnitz."
DISPLAY_NAME	The name to display when referencing the IP component, such as "My Qsys Pro IP Component".
EDITABLE	Indicates whether you can edit the IP component in the Component Editor.
ELABORATION_CALLBACK	The name of the elaboration callback. When set, the IP component's elaboration callback is called to validate and elaborate interfaces for instances of the IP component.
GENERATION_CALLBACK	The name for a custom generation callback.
GROUP	The group in the IP Catalog that includes this IP component.
ICON_PATH	A path to an icon to display in the IP component's parameter editor.
INSTANTIATE_IN_SYSTEM_MODULE	If true, this IP component is implemented by HDL provided by the IP component. If false, the IP component will create exported interfaces allowing the implementation to be connected in the parent.
INTERNAL	An IP component which is marked as internal does not appear in the IP Catalog. This feature allows you to hide the sub-IP-components of a larger composed IP component.
MODULE_DIRECTORY	The directory in which the hw.tcl file exists.
MODULE_TCL_FILE	The path to the hw.tcl file.
NAME	The name of the IP component, such as `my_qsys_component`.
OPAQUE_ADDRESS_MAP	For composed IP components created using a _hw.tcl file that include children that are memory-mapped slaves, specifies whether the children's addresses are visible to downstream software tools. When `true`, the children's address are not visible. When `false`, the children's addresses are visible.
PREFERRED_SIMULATION_LANGUAGE	The preferred language to use for selecting the fileset for simulation model generation.
REPORT_HIERARCHY	null
STATIC_TOP_LEVEL_MODULE_NAME	Deprecated.

Name	Description
STRUCTURAL_COMPOSITION_CALLBACK	The name of the structural composition callback. This callback is used to represent the structural hierarchical model of the IP component and the RTL can be generated either with module property COMPOSITION_CALLBACK or by ADD_FILESET with target QUARTUS_SYNTH
SUPPORTED_DEVICE_FAMILIES	A list of device family supported by this IP component.
TOP_LEVEL_HDL_FILE	Deprecated.
TOP_LEVEL_HDL_MODULE	Deprecated.
UPGRADEABLE_FROM	null
VALIDATION_CALLBACK	The name of the validation callback procedure.
VERSION	The IP component's version, such as 10.0.

13.2.13 Fileset Properties

Name	Description
ENABLE_FILE_OVERWRITE_MODE	null
ENABLE_RELATIVE_INCLUDE_PATHS	If true, HDL files can include other files using relative paths in the fileset.
TOP_LEVEL	The name of the top-level HDL module that the fileset generates. If set, the HDL top level must match the TOP_LEVEL name, and the HDL must not be parameterized. Qsys Pro runs the generate callback one time, regardless of the number of instances in the system.



13.2.14 Fileset Kind Properties

Name	Description
EXAMPLE_DESIGN	Contains example design files.
QUARTUS_SYNTH	Contains files that Qsys Pro uses for the Quartus Prime software synthesis.
SIM_VERILOG	Contains files that Qsys Pro uses for Verilog HDL simulation.
SIM_VHDL	Contains files that Qsys Pro uses for VHDL simulation.

13.2.15 Callback Properties

Description

This list describes each type of callback. Each command may only be available in some callback contexts.

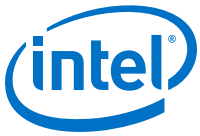
Name	Description
ACTION	Called when an ACTION display item's action is performed.
COMPOSITION	Called during instance elaboration when the IP component contains a subsystem.
EDITOR	Called when the IP component is controlling the parameterization editor.
ELABORATION	Called to elaborate interfaces and signals after a parameter change. In API 9.1 and later, validation is called before elaboration. In API 9.0 and earlier, elaboration is called before validation.
GENERATE_VERILOG_SIMULATION	Called when the IP component uses a custom generator to generates the Verilog simulation model for an instance.
GENERATE_VHDL_SIMULATION	Called when the IP component uses a custom generator to generates the VHDL simulation model for an instance.
GENERATION	Called when the IP component uses a custom generator to generates the synthesis HDL for an instance.
PARAMETER_UPGRADE	Called when attempting to instantiate an IP component with a newer version than the saved version. This allows the IP component to upgrade parameters between released versions of the component.
STRUCTURAL_COMPOSITION	Called during instance elaboration when an IP component is represented by a structural hierarchical model which may be different from the generated RTL.
VALIDATION	Called to validate parameter ranges and report problems with the parameter values. In API 9.1 and later, validation is called before elaboration. In API 9.0 and earlier, elaboration is called before validation.

13.2.16 File Attribute Properties

Name	Description
ALDEC_SPECIFIC	Applies to Aldec simulation tools and for simulation filesets only.
CADENCE_SPECIFIC	Applies to Cadence simulation tools and for simulation filesets only.
COMMON_SYSTEMVERILOG_PACKAGE	The name of the common SystemVerilog package. Applies to simulation filesets only.
MENTOR_SPECIFIC	Applies to Mentor simulation tools and for simulation filesets only.
SYNOPSYS_SPECIFIC	Applies to Synopsys simulation tools and for simulation filesets only.
TOP_LEVEL_FILE	Contains the top-level module for the fileset and applies to synthesis filesets only.

13.2.17 File Kind Properties

Name	Description
DAT	DAT Data
FLI_LIBRARY	FLI Library
HEX	HEX Data
MIF	MIF Data
OTHER	Other
PLI_LIBRARY	PLI Library
SDC	Timing Constraints
SYSTEM_VERILOG	SystemVerilog HDL
SYSTEM_VERILOG_ENCRYPT	Encrypted SystemVerilog HDL
SYSTEM_VERILOG_INCLUDE	SystemVerilog Include
VERILOG	Verilog HDL
VERILOG_ENCRYPT	Encrypted Verilog HDL
VERILOG_INCLUDE	Verilog Include
VHDL	VHDL
VHDL_ENCRYPT	Encrypted VHDL
VPI_LIBRARY	VPI Library

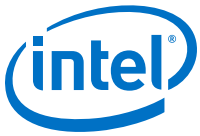


13.2.18 File Source Properties

Name	Description
PATH	Specifies the original source file and copies to `output_file`.
TEXT	Specifies an arbitrary text string for the contents of `output_file`.

13.2.19 Simulator Properties

Name	Description
ENV_ALDEC_LD_LIBRARY_PATH	LD_LIBRARY_PATH when running riviera-pro
ENV_CADENCE_LD_LIBRARY_PATH	LD_LIBRARY_PATH when running ncsim
ENV_MENTOR_LD_LIBRARY_PATH	LD_LIBRARY_PATH when running modelsim
ENV_SYNOPSYS_LD_LIBRARY_PATH	LD_LIBRARY_PATH when running vcs
OPT_ALDEC_PLI	-pli option for riviera-pro
OPT_CADENCE_64BIT	-64bit option for ncsim
OPT_CADENCE_PLI	-loadpli1 option for ncsim
OPT_CADENCE_SVLIB	-sv_lib option for ncsim
OPT_CADENCE_SVROOT	-sv_root option for ncsim
OPT_MENTOR_64	-64 option for modelsim
OPT_MENTOR_CPPPATH	-cpppath option for modelsim
OPT_MENTOR_LDFLAGS	-ldflags option for modelsim
OPT_MENTOR_PLI	-pli option for modelsim
OPT_SYNOPSYS_ACC	+acc option for vcs
OPT_SYNOPSYS_CPP	-cpp option for vcs
OPT_SYNOPSYS_FULL64	-full64 option for vcs
OPT_SYNOPSYS_LDFLAGS	-LDFLAGS option for vcs
OPT_SYNOPSYS_LLIB	-l option for vcs
OPT_SYNOPSYS_VPI	+vpi option for vcs



13.2.20 Port VHDL Type Properties

Name	Description
AUTO	The VHDL type of this signal is automatically determined. Single-bit signals are STD_LOGIC; signals wider than one bit will be STD_LOGIC_VECTOR.
STD_LOGIC	Indicates that the signal in not rendered in VHDL as a STD_LOGIC signal.
STD_LOGIC_VECTOR	Indicates that the signal is rendered in VHDL as a STD_LOGIC_VECTOR signal.

13.2.21 System Info Type Properties

Type	Name	Description
String	ADDRESS_MAP	An XML-formatted string describing the address map for the interface specified in the system info argument.
Integer	ADDRESS_WIDTH	The number of address bits required to address all memory-mapped slaves connected to the specified memory-mapped master in this instance, using byte addresses.
String	AVALON_SPEC	The version of the interconnect. SOPC Builder interconnect uses Avalon Specification 1.0. Qsys Pro interconnect uses Avalon Specification 2.0.
Integer	CLOCK_DOMAIN	An integer that represents the clock domain for the interface specified in the system info argument. If this instance has interfaces on multiple clock domains, this can be used to determine which interfaces are on each clock domain. The absolute value of the integer is arbitrary.
Long, Integer	CLOCK_RATE	The rate of the clock connected to the clock input specified in the system info argument. If 0, the clock rate is currently unknown.
String	CLOCK_RESET_INFO	The name of this instance's primary clock or reset sink interface. This is used to determine the reset sink to use for global reset when using SOPC interconnect.
String	CUSTOM_INSTRUCTION_SLAVES	Provides custom instruction slave information, including the name, base address, address span, and clock cycle type.
(various)	DESIGN_ENVIRONMENT	A string that identifies the current design environment. Refer to *Design Environment Type Properties*.
String	DEVICE	The device part number of the currently selected device.
String	DEVICE_FAMILY	The family name of the currently selected device.
String	DEVICE_FEATURES	A list of key/value pairs delineated by spaces indicating whether a particular device feature is available in the currently selected device family. The format of the list is suitable for passing to the Tcl `array` set command. The keys are device features; the values will be 1 if the feature is present, and 0 if the feature is absent.
String	DEVICE_SPEEDGRADE	The speed grade of the currently selected device.
Integer	GENERATION_ID	A integer that stores a hash of the generation time to be used as a unique ID for a generation run.
BigInteger, Long	INTERRUPTS_USED	A mask indicating which bits of an interrupt receiver are connected to interrupt senders. The interrupt receiver is specified in the system info argument.
Integer	MAX_SLAVE_DATA_WIDTH	The data width of the widest slave connected to the specified memory-mapped master.
String, Boolean, Integer	QUARTUS_INI	The value of the quartus.ini setting specified in the system info argument.
Integer	RESET_DOMAIN	An integer that represents the reset domain for the interface specified in the system info argument. If this instance has interfaces on multiple reset domains, this can be used to determine which interfaces are on each reset domain. The absolute value of the integer is arbitrary.

Type	Name	Description
String	TRISTATECONDUIT_INFO	An XML description of the Avalon Tri-state Conduit masters connected to an Avalon Tri-state Conduit slave. The slave is specified as the system info argument. The value will contain information about the slave, connected master instance and interface names, and signal names, directions and widths.
String	TRISTATECONDUIT_MASTERS	The names of the instance's interfaces that are tri-state conduit slaves.
String	UNIQUE_ID	A string guaranteed to be unique to this instance.

Related Links

Design Environment Type Properties on page 874

13.2.22 Design Environment Type Properties

Description

A design environment is used by IP to tell what sort of interfaces are most appropriate to connect in the parent system.

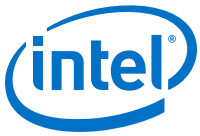
Name	Description
NATIVE	Design environment prefers native IP interfaces.
QSYS	Design environment prefers standard Qsys Pro interfaces.

13.2.23 Units Properties

Name	Description
Address	A memory-mapped address.
Bits	Memory size, in bits.
BitsPerSecond	Rate, in bits per second.
Bytes	Memory size, in bytes.
Cycles	A latency or count, in clock cycles.
GigabitsPerSecond	Rate, in gigabits per second.
Gigabytes	Memory size, in gigabytes.
Gigahertz	Frequency, in GHz.
Hertz	Frequency, in Hz.
KilobitsPerSecond	Rate, in kilobits per second.
Kilobytes	Memory size, in kilobytes.
Kilohertz	Frequency, in kHz.
MegabitsPerSecond	Rate, in megabits per second.
Megabytes	Memory size, in megabytes.
Megahertz	Frequency, in MHz.
Microseconds	Time, in micros.
Milliseconds	Time, in ms.
Nanoseconds	Time, in ns.
None	Unspecified units.
Percent	A percentage.
Picoseconds	Time, in ps.
Seconds	Time, in s.

13.2.24 Operating System Properties

Name	Description
ALL	All operating systems
LINUX32	Linux 32-bit
LINUX64	Linux 64-bit
WINDOWS32	Windows 32-bit
WINDOWS64	Windows 64-bit



13.2.25 Quartus.ini Type Properties

Name	Description
ENABLED	Returns `1` if the setting is turned on, otherwise returns `0`.
STRING	Returns the string value of the `.ini` setting.

13.3 Document Revision History

The table below indicates edits made to the *Component Interface Tcl Reference* content since its creation.

Date	Version	Changes
2016.10.31	16.1.0	• Implemented Intel rebranding. • Implemented Qsys Pro rebranding.
2015.11.02	15.1.0	Changed instances of *Quartus II* to *Quartus Prime*.
2015.05.04	15.0.0	Edit to `add_fileset_file` command.
December 2014	14.1.0	• set_interface_upgrade_map • Moved **Port Roles (Interface Signal Types)** section to *Qsys Pro Interconnect*.
November 2013	13.1.0	• add_hdl_instance
May 2013	13.0.0	• Consolidated content from other Qsys Pro chapters. • Added AMBA APB support.
November 2012	12.1.0	• Added the **demo_axi_memory** example with screen shots and example **_hw.tcl** code.
June 2012	12.0.0	• Added AMBA AXI3 support. • Added: `set_display_item_property`, `set_parameter_property`,`LONG_DESCRIPTION`, and static filesets.
November 2011	11.1.0	• Template update. • Added: `set_qip_strings`, `get_qip_strings`, `get_device_family_displayname`, `check_device_family_equivalence`.
May 2011	11.0.0	• Revised section describing HDL and composed component implementations. • Separated reset and clock interfaces in example. • Added: `TRISTATECONDUIT_INFO`, `GENERATION_ID`, `UNIQUE_ID` `SYSTEM_INFO`. • Added: `WIDTH` and `SYSTEM_INFO_ARG` parameter properties. • Removed the `doc_type` argument from the `add_documentation_link` command. • Removed: `get_instance_parameter_properties` • Added: `add_fileset`, `add_fileset_file`, `create_temp_file`. • Updated Tcl examples to show separate clock and reset interfaces.
December 2010	10.1.0	• Initial release.

Related Links

Altera Documentation Archive

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

14 Qsys Pro System Design Components

You can use Qsys Pro IP components to create Qsys Pro systems. Qsys Pro interfaces include components appropriate for streaming high-speed data, reading and writing registers and memory, controlling off-chip devices, and transporting data between components.

Qsys Pro supports Avalon, AMBA AXI3 (version 1.0), AMBA AXI4 (version 2.0), AMBA AXI4-Lite (version 2.0), AMBA AXI4-Stream (version 1.0), and AMBA APB3 (version 1.0) interface specifications.

Related Links

- Creating a System With Qsys Pro on page 299
  Qsys Pro is a system integration tool included as part of the Quartus Prime software.
- Qsys Pro Interconnect on page 627
  Qsys Pro interconnect is a high-bandwidth structure that allows you to connect IP components to other IP components with various interfaces.
- Avalon Interface Specifications
- AMBA Protocol Specifications
- Embedded Peripherals IP User Guide

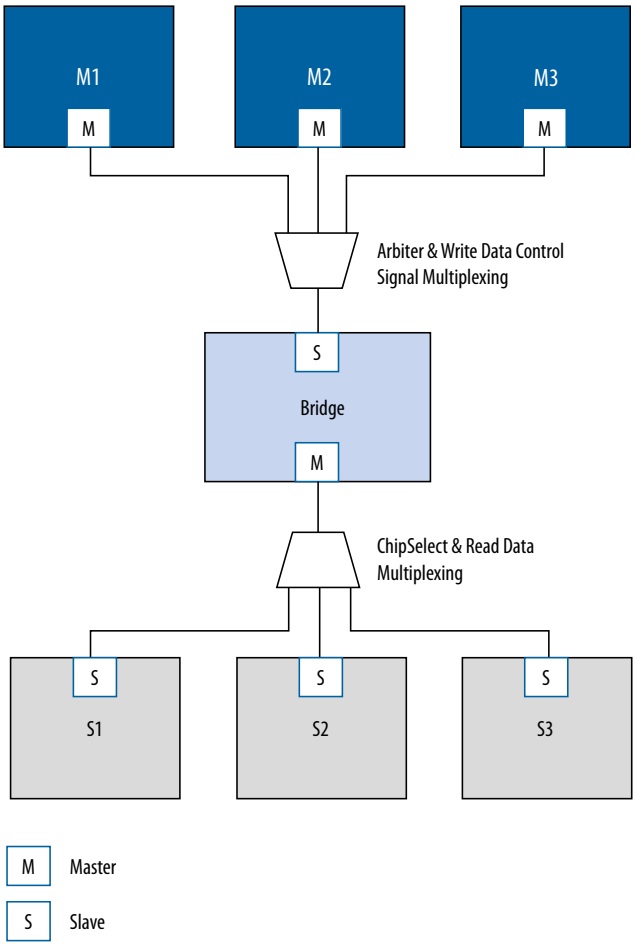
14.1 Bridges

Bridges affect the way Qsys Pro transports data between components. You can insert bridges between master and slave interfaces to control the topology of a Qsys Pro system, which affects the interconnect that Qsys Pro generates. You can also use bridges to separate components into different clock domains to isolate clock domain crossing logic.

A bridge has one slave interface and one master interface. In Qsys Pro, one or more master interfaces from other components connect to the bridge slave. The bridge master connects to one or more slave interfaces on other components.

Figure 245. Using a Bridge in a Qsys Pro System

In this example, three masters have logical connections to three slaves, although physically each master connects only to the bridge. Transfers initiated to the slave propagate to the master in the same order in which the transfers are initiated on the slave.

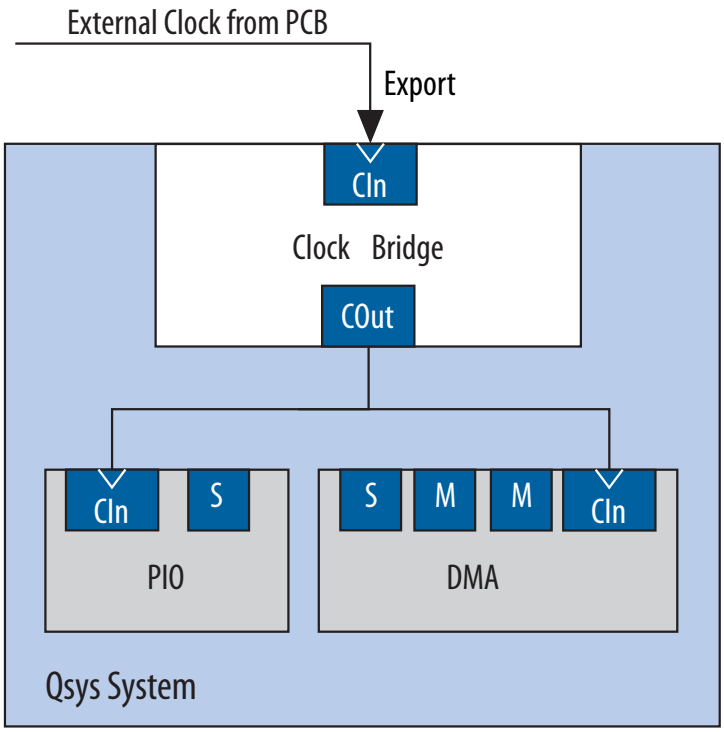


14.1.1 Clock Bridge

The Clock Bridge connects a clock source to multiple clock input interfaces. You can use the clock bridge to connect a clock source that is outside the Qsys Pro system. Create the connection through an exported interface, and then connect to multiple clock input interfaces.

Clock outputs match fan-out performance without the use of a bridge. You require a bridge only when you want a clock from an exported source to connect internally to more than one source.

Figure 246. Clock Bridge



14.1.2 Avalon-MM Clock Crossing Bridge

The Avalon-MM Clock Crossing Bridge transfers Avalon-MM commands and responses between different clock domains. You can also use the Avalon-MM Clock Crossing Bridge between AXI masters and slaves of different clock domains.

The Avalon-MM Clock Crossing Bridge uses asynchronous FIFOs to implement clock crossing logic. The bridge parameters control the depth of the command and response FIFOs in both the master and slave clock domains. If the number of active reads exceeds the depth of the response FIFO, the Clock Crossing Bridge stops sending reads.

To maintain throughput for high-performance applications, increase the response FIFO depth from the default minimum depth, which is twice the maximum burst size.

Note: When you use the FIFO-based clock crossing a Qsys Pro system, the DC FIFO is automatically inserted in the Qsys Pro system. The reset inputs for the DC FIFO connect to the reset sources for the connected master and slave components on either side of the DC FIFO. For this configuration, you must assert both the resets on the master and the slave sides at the same time to ensure the DC FIFO resets properly. Alternatively, you can drive both resets from the same reset source to guarantee that the DC FIFO resets properly.

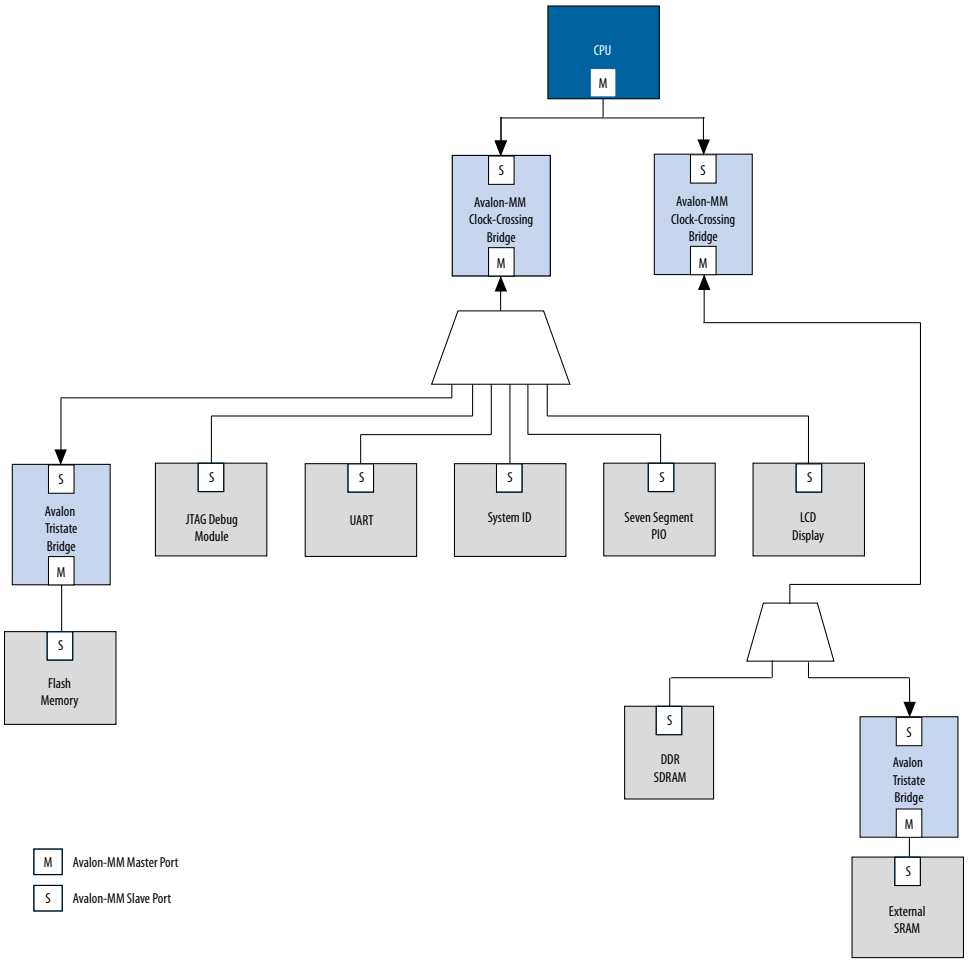
Note: The clock crossing bridge includes appropriate SDC constraints for its internal asynchronous FIFOs. For these SDC constraints to work correctly, do not set false paths on the pointer crossings in the FIFOs. You should also not split the bridge's clocks into separate clock groups when you declare SDC constraints; the split has the same effect as setting false paths.

14.1.2.1 Avalon-MM Clock Crossing Bridge Example

In the example shown below, the Avalon-MM Clock Crossing bridges separate slave components into two groups. The Avalon-MM Clock Crossing Bridge places the low performance slave components behind a single bridge and clocks the components at a lower speed. The bridge places the high performance components behind a second bridge and clocks it at a higher speed.

By inserting clock-crossing bridges, you simplify the Qsys Pro interconnect and allow the Quartus Prime Fitter to optimize paths that require minimal propagation delay.

Figure 247. Avalon-MM Clock Crossing Bridge



14.1.2.2 Avalon-MM Clock Crossing Bridge Parameters

Table 179. Avalon-MM Clock Crossing Bridge Parameters

Parameters	Values	Description
Data width	8, 16, 32, 64, 128, 256, 512, 1024 bits	Determines the data width of the interfaces on the bridge, and affects the size of both FIFOs. For the highest bandwidth, set **Data width** to be as wide as the widest master that connects to the bridge.
Symbol width	1, 2, 4, 8, 16, 32, 64 (bits)	Number of bits per symbol. For example, byte-oriented interfaces have 8-bit symbols.
Address width	1-32 bits	The address bits needed to address the downstream slaves.
Use automatically-determined address width	-	The minimum bridge address width that is required to address the downstream slaves.
Maximum burst size	1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024 bits	Determines the maximum length of bursts that the bridge supports.
Command FIFO depth	2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384 bits	Command (master-to-slave) FIFO depth.
Respond FIFO depth	2, 4, 8,16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384 bits	Slave-to-master FIFO depth.
Master clock domain synchronizer depth	2, 3, 4, 5 bits	The number of pipeline stages in the clock crossing logic in the issuing master to target slave direction. Increasing this value leads to a larger mean time between failures (MTBF). You can determine the MTBF for a design by running a TimeQuest timing analysis.
Slave clock domain synchronizer depth	2, 3, 4, 5 bits	The number of pipeline stages in the clock crossing logic in the target slave to master direction. Increasing this value leads to a larger meantime between failures (MTBF). You can determine the MTBF for a design by running a TimeQuest timing analysis.

14.1.3 Avalon-MM Pipeline Bridge

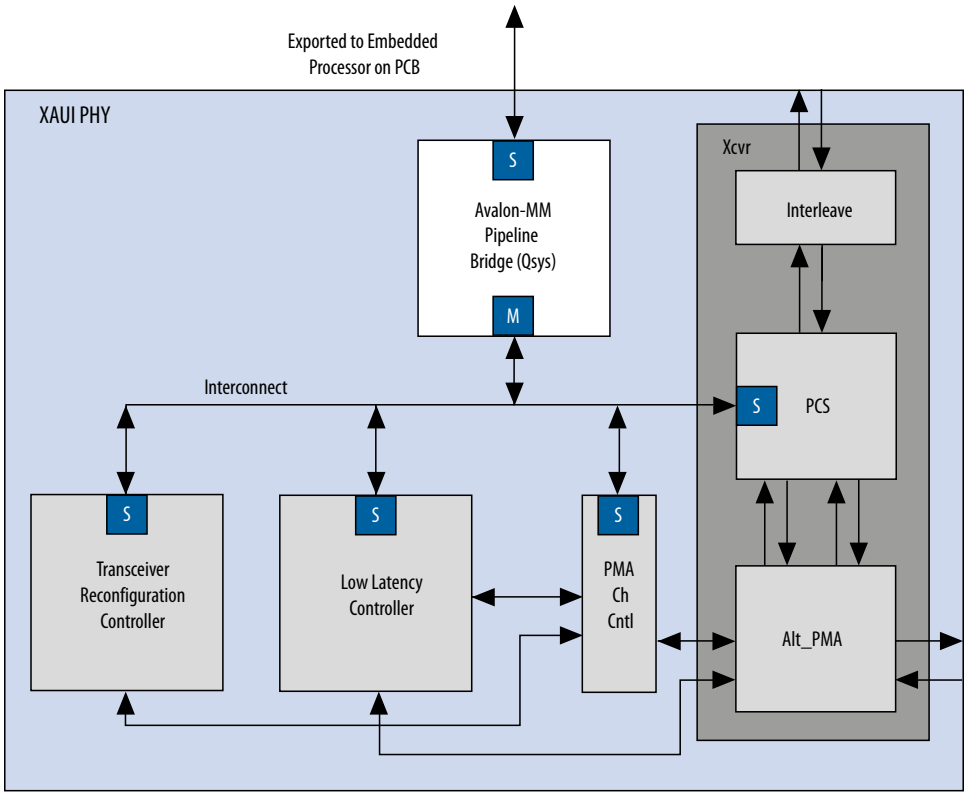
The Avalon-MM Pipeline Bridge inserts a register stage in the Avalon-MM command and response paths. The bridge accepts commands on its slave port and propagates the commands to its master port. The pipeline bridge provides separate parameters to turn on pipelining for command and response signals.

The **Maximum pending read transactions** parameter is the maximum number of pending reads that the Avalon-MM bridge can queue up. To determine the best value for this parameter, review this same option for the bridge's connected slaves and identify the highest value of the parameter, and then add the internal buffering requirements of the Avalon-MM bridge. In general, the value should be between 4 and 32. The limit for maximum queued transactions is 64.

You can use the Avalon-MM bridge to export a single Avalon-MM slave interface to control multiple Avalon-MM slave devices. The pipelining feature is optional.

Figure 248. Avalon-MM Pipeline Bridge in a XAUI PHY Transceiver IP Core

In this example, the bridge transfers commands received on its slave interface to its master port.

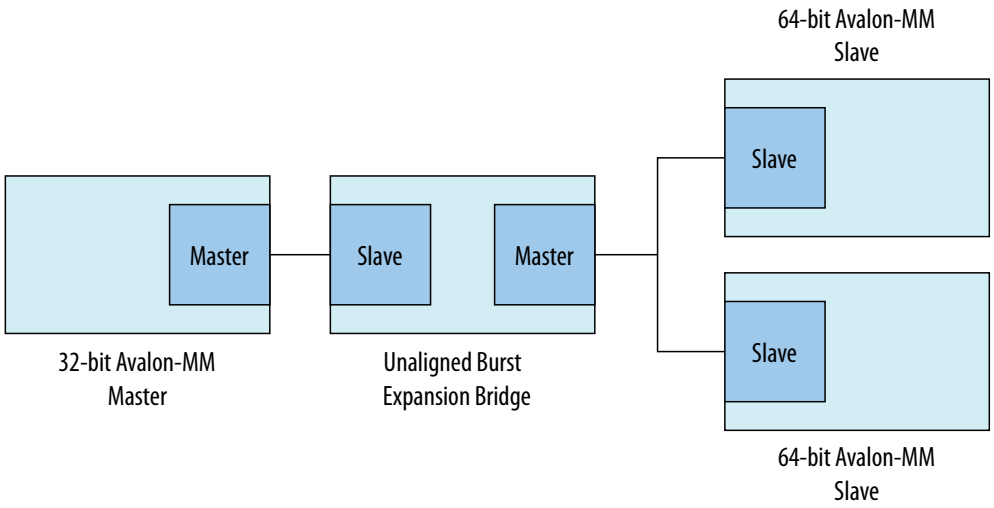


Because the slave interface is exported to the pins of the device, having a single slave port, rather than separate ports for each slave device, reduces the pin count of the FPGA.

14.1.4 Avalon-MM Unaligned Burst Expansion Bridge

The Avalon-MM Unaligned Burst Expansion Bridge aligns read burst transactions from masters connected to its slave interface, to the address space of slaves connected to its master interface. This alignment ensures that all read burst transactions are delivered to the slave as a single transaction.

Figure 249. Avalon-MM Unaligned Burst Expansion Bridge



You can use the Avalon Unaligned Burst Expansion Bridge to align read burst transactions from masters that have narrower data widths than the target slaves. Using the bridge for this purpose improves bandwidth utilization for the master-slave pair, and ensures that unaligned bursts are processed as single transactions rather than multiple transactions.

Note: Do not use the Avalon-MM Unaligned Burst Expansion Bridge if any connected slave has read side effects from reading addresses that are exposed to any connected master's address map. This bridge can cause read side effects due to alignment modification to read burst transaction addresses.

Note: The Avalon-MM Unaligned Burst Expansion Bridge does not support VHDL simulation.

Related Links

Qsys Pro Interconnect on page 627
Qsys Pro interconnect is a high-bandwidth structure that allows you to connect IP components to other IP components with various interfaces.

14.1.4.1 Using the Avalon-MM Unaligned Burst Expansion Bridge

When a master sends a read burst transaction to a slave, the Avalon-MM Unaligned Burst Expansion Bridge initially determines whether the start address of the read burst transaction is aligned to the slave's memory address space. If the base address is aligned, the bridge does not change the base address. If the base address is not aligned, the bridge aligns the base address to the nearest aligned address that is less than the requested base address.

The Avalon-MM Unaligned Burst Expansion Bridge then determines whether the final word requested by the master is the last word at the slave read burst address. If a single slave address contains multiple words, all of those words must be requested in order for a single read burst transaction to occur.

- If the final word requested by the master is the last word at the slave read burst address, the bridge does not modify the burst length of the read burst command to the slave.
- If the final word requested by the master is not the last word at the slave read burst address, the bridge increases the burst length of the read burst command to the slave. The final word requested by the modified read burst command is then the last word at the slave read burst address.

The bridge stores information about each aligned read burst command that it sends to slaves connected to a master interface. When a read response is received on the master interface, the bridge determines if the base address or burst length of the issued read burst command was altered.

If the bridge alters either the base address or the burst length of the issued read burst command, it receives response words that the master did not request. The bridge suppresses words that it receives from the aligned burst response that are not part of the original read burst command from the master.

14.1.4.2 Avalon-MM Unaligned Burst Expansion Bridge Parameters

Figure 250. Avalon-MM Unaligned Burst Expansion Bridge Parameter Editor

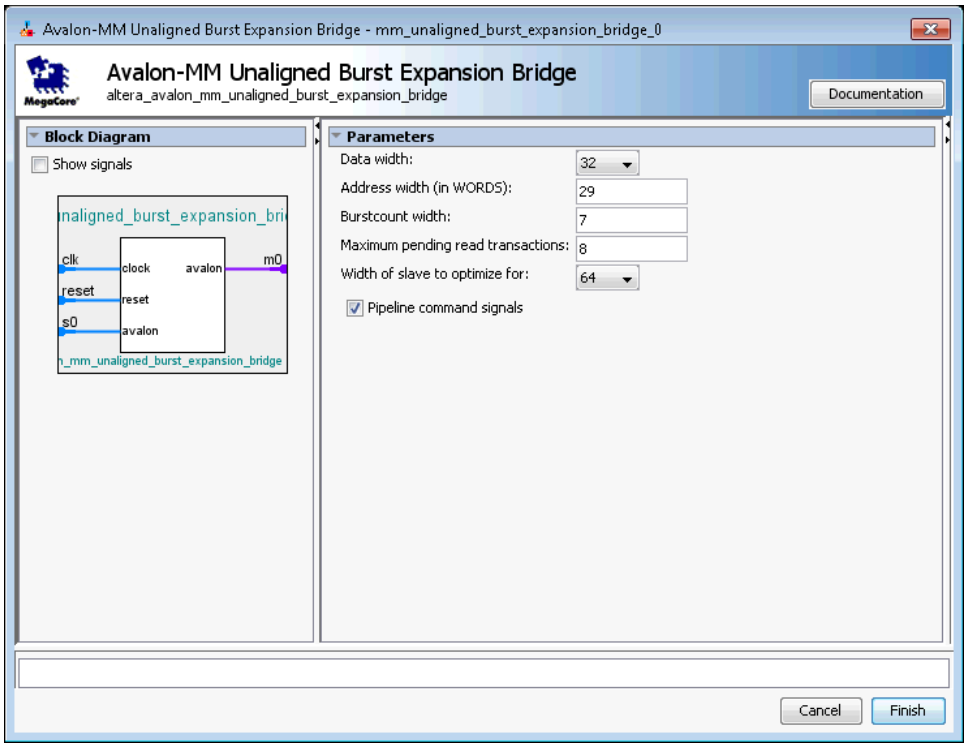


Table 180. Avalon-MM Unaligned Burst Expansion Bridge Parameters

Parameter	Description
Data width	Data width of the master connected to the bridge.
Address width (in WORDS)	The address width of the master connected to the bridge.

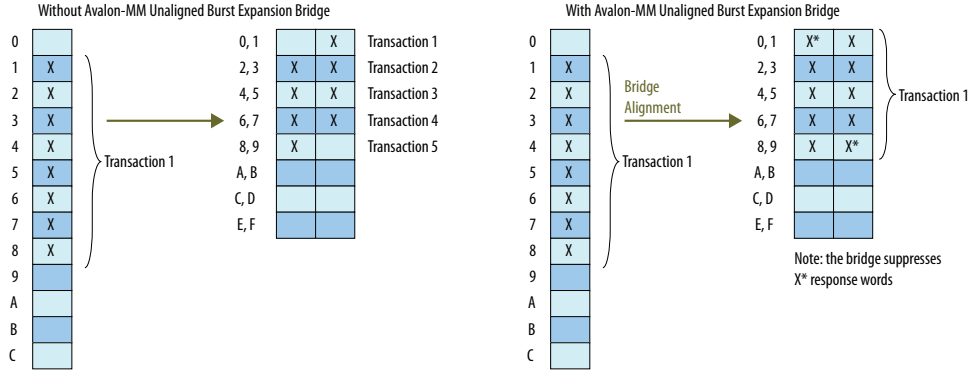
continued...

Parameter	Description
Burstcount width	The burstcount signal width of the master connected to the bridge.
Maximum pending read transactions	The **Maximum pending read transactions** parameter is the maximum number of pending reads that the Avalon-MM bridge can queue up. To determine the best value for this parameter, review this same option for the bridge's connected slaves and identify the highest value of the parameter, and then add the internal buffering requirements of the Avalon-MM bridge. In general, the value should be between 4 and 32. The limit for maximum queued transactions is 64.
Width of slave to optimize for	The data width of the connected slave. Supported values are: 16, 32, 64, 128, 256, 512, 1024, 2048, and 4096 bits. *Note:* If you connect multiple slaves, all slaves must have the same data width.
Pipeline command signals	When turned on, the command path is pipelined, minimizing the bridge's critical path at the expense of increased logic usage and latency.

14.1.4.3 Avalon-MM Unaligned Burst Expansion Bridge Example

Figure 251. Unaligned Burst Expansion Bridge

The example below shows an unaligned read burst command from a master that the Avalon-MM Unaligned Burst Expansion Bridge converts to an aligned request for a connected slave, and the suppression of words due to the aligned read burst command. In this example, a 32-bit master requests an 8-beat burst of 32-bit words from a 64-bit slave with a start address that is not 64-bit aligned.



Because the target slave has a 64-bit data width, address 1 is unaligned in the slave's address space. As a result, several smaller burst transactions are needed to request the data associated with the master's read burst command.

With an Avalon-MM Unaligned Burst Expansion Bridge in place, the bridge issues a new read burst command to the target slave beginning at address 0 with burst length 10, which requests data up to the word stored at address 9.

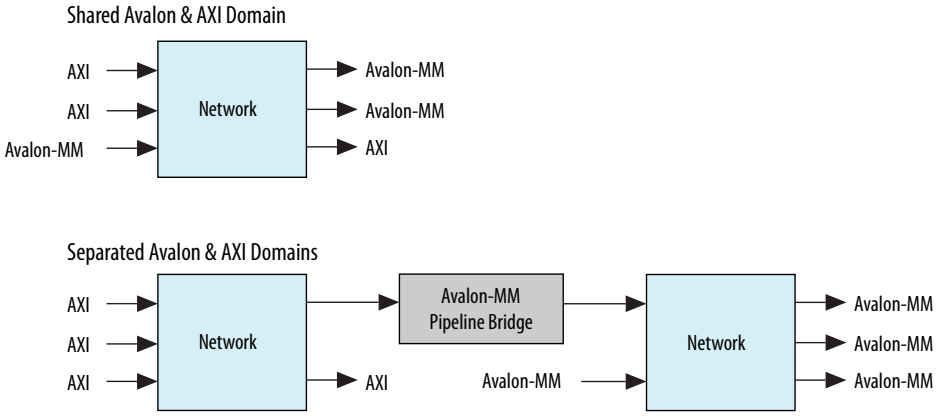
When the bridge receives the word corresponding to address 0, it suppresses it from the master, and then delivers the words corresponding to addresses 1 through 8 to the master. When the bridge receives the word corresponding to address 9, it suppresses that word from the master.

14.1.5 Bridges Between Avalon and AXI Interfaces

When designing a Qsys Pro system, you can make connections between AXI and Avalon interfaces without the use of explicitly-instantiated bridges; the interconnect provides all necessary bridging logic. However, this does not prevent the use of explicit bridges to separate the AXI and Avalon domains.

Figure 252. Avalon-MM Pipeline Bridge Between Avalon-MM and AXI Domains

Using an explicit Avalon-MM bridge to separate the AXI and Avalon domains reduces the amount of bridging logic in the interconnect at the expense of concurrency.



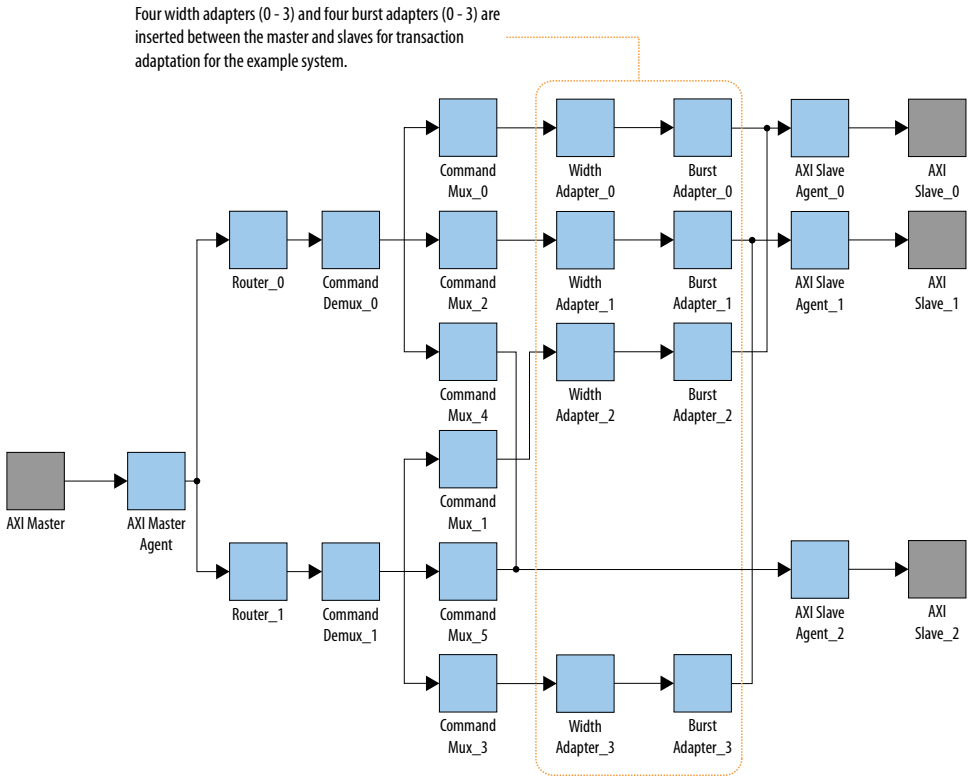
14.1.6 AXI Bridge

With an AXI bridge, you can influence the placement of resource-intensive components, such as the width and burst adapters. Depending on its use, an AXI bridge may reduce throughput and concurrency, in return for higher f_{MAX} and less logic.

You can use an AXI bridge to group different parts of your Qsys Pro system. Other parts of the system can then connect to the bridge interface instead of to multiple separate master or slave interfaces. You can also use an AXI bridge to export AXI interfaces from Qsys Pro systems.

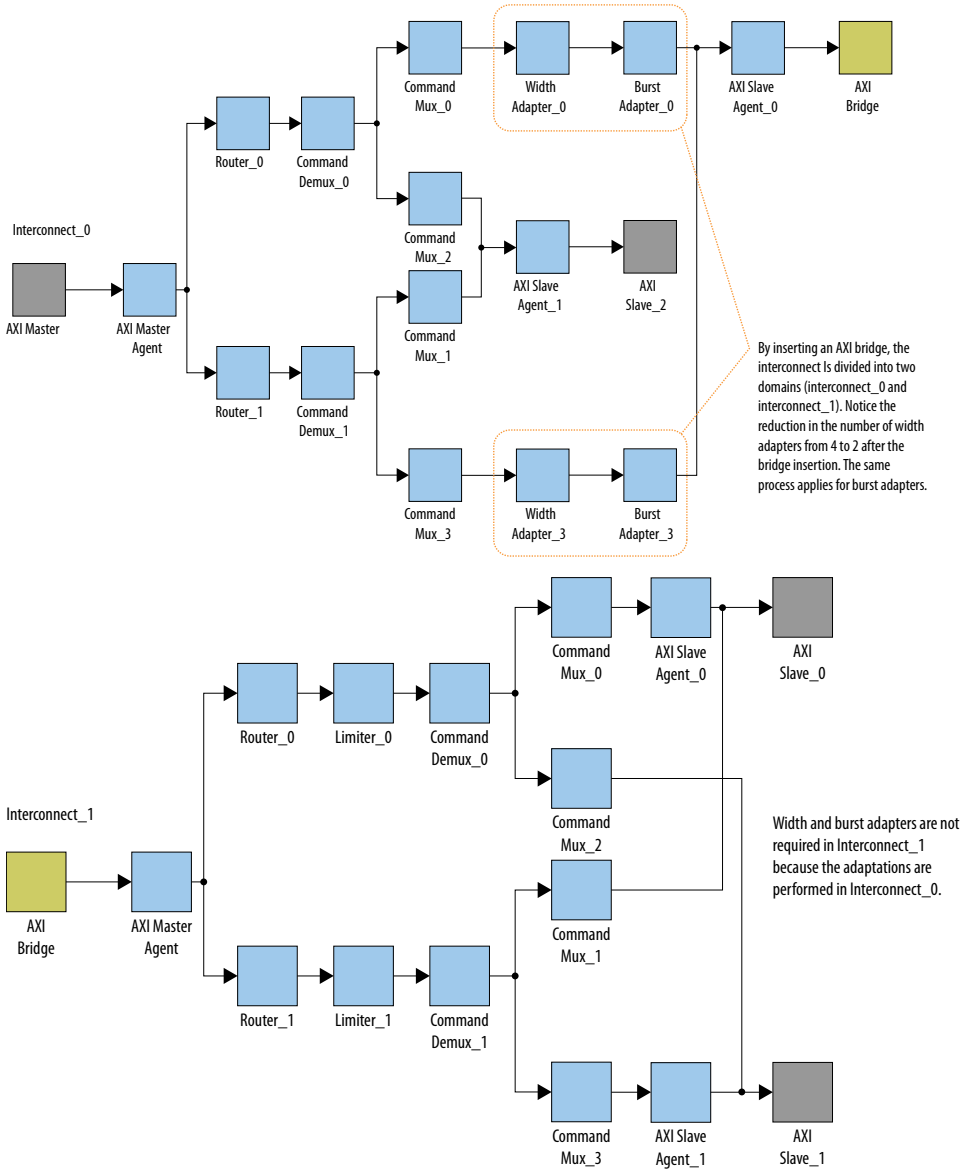
The example below shows a system with a single AXI master and three AXI slaves. It also has various interconnect components, such as routers, demultiplexers, and multiplexers. Two of the slaves have a narrower data width than the master; 16-bit slaves versus a 32-bit master. In this system, Qsys Pro interconnect creates four width adapters and four burst adapters to access the two slaves. You could improve resource usage by adding an AXI bridge. Then, Qsys Pro needs to add only two width adapters and two burst adapters; one pair for the read channels, and another pair for the write channel.

Figure 253. AXI Example Without a Bridge: Adding a Bridge Can Reduce the Number of Adapters



The example below shows the same system with an AXI bridge component, and the decrease in the number of width and burst adapters. Qsys Pro creates only two width adapters and two burst adapters, as compared to the four width adapters and four burst adapters in the previous example. The system includes more components, but the overall system performance improves because there are fewer resource-intensive width and burst adapters.

Figure 254. Width and Burst Adapters Added to a System With a Bridge



14.1.6.1 AXI Bridge Signal Types

Based on parameter selections that you make for the AXI Bridge component, Qsys Pro instantiates either the AXI3 or AXI4 master and slave interfaces into the component.

Note: In AXI3, `aw/aruser` accommodates sideband signal usage by hard processor systems (HPS).

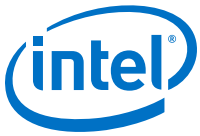


Table 181. Sets of Signals for the AXI Bridge Based on the Protocol

Signal Name	AXI3	AXI4
awid / arid	yes	yes
awaddr / araddr	yes	yes
awlen / arlen	yes (4-bit)	yes (8-bit)
awsize / arsize	yes	yes
awburst / arburst	yes	yes
awlock / arlock	yes	yes (1-bit optional)
awcache / arcache	yes (2-bit)	yes (optional)
awprot / arprot	yes	yes
awuser / aruser	yes	yes
awvalid / arvalid	yes	yes
awready / arready	yes	yes
awqos / arqos	no	yes
awregion / arregion	no	yes
wid	yes	no (optional)
wdata / rdata	yes	yes
wstrb	yes	yes
wlast / rvalid	yes	yes
wvalid / rlast	yes	yes
wready / rready	yes	yes
wuser / ruser	no	yes
bid / rid	yes	yes
bresp / rresp	yes	yes (optional)
bvalid	yes	yes
bready	yes	yes

14.1.6.2 AXI Bridge Parameters

In the parameter editor, you can customize the parameters for the AXI bridge according to the requirements of your design.

Figure 255. AXI Bridge Parameter Editor

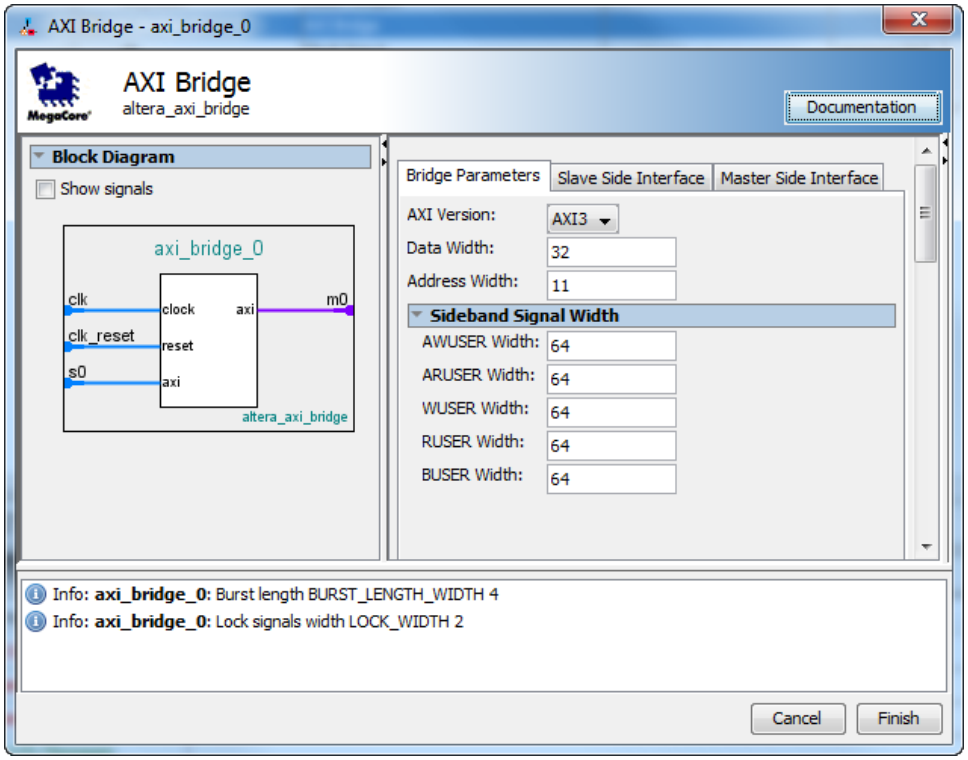


Table 182. AXI Bridge Parameters

Parameter	Type	Range	Description
AXI Version	string	AXI3/ AXI4	Specifies the AXI version and signals that Qsys Pro generates for the slave and master interfaces of the bridge.
Data Width	integer	8:1024	Controls the width of the data for the master and slave interfaces.
Address Width	integer	1-64 bits	Controls the width of the address for the master and slave interfaces.
AWUSER Width	integer	1-64 bits	Controls the width of the write address channel sideband signals of the master and slave interfaces.
ARUSER Width	integer	1-64 bits	Controls the width of the read address channel sideband signals of the master and slave interfaces.
WUSER Width	integer	1-64 bits	Controls the width of the write data channel sideband signals of the master and slave interfaces.
RUSER Width	integer	1-16 bits	Controls the width of the read data channel sideband signals of the master and slave interfaces.
BUSER Width	integer	1-16 bits	Controls the width of the write response channel sideband signals of the master and slave interfaces.

14.1.6.3 AXI Bridge Slave and Master Interface Parameters

Table 183. AXI Bridge Slave and Master Interface Parameters

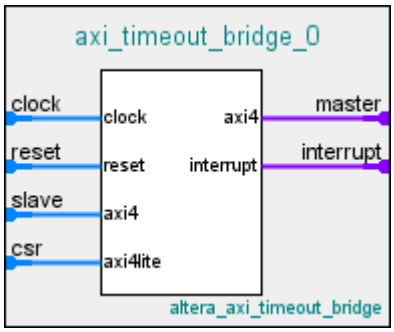
Parameter	Description
ID Width	Controls the width of the thread ID of the master and slave interfaces.
Write/Read/Combined Acceptance Capability	Controls the depth of the FIFO that Qsys Pro needs in the interconnect agents based on the maximum pending commands that the slave interface accepts.
Write/Read/Combined Issuing Capability	Controls the depth of the FIFO that Qsys Pro needs in the interconnect agents based on the maximum pending commands that the master interface issues. Issuing capability must follow acceptance capability to avoid unnecessary creation of FIFOs in the bridge.

Note: Maximum acceptance/issuing capability is a model-only parameter and does not influence the bridge HDL. The bridge does not backpressure when this limit is reached. Downstream components and/or the interconnect must apply backpressure.

14.1.7 AXI Timeout Bridge

You can place an AXI Timeout Bridge between a single master and a single slave if you know that the slave may time out and cause your system to hang. If a slave does not accept a command or respond to a command it accepted, its master can wait indefinitely. The AXI Timeout Bridge allows your system to recover when it hangs, and also facilitates debugging.

Figure 256. AXI Timeout Bridge

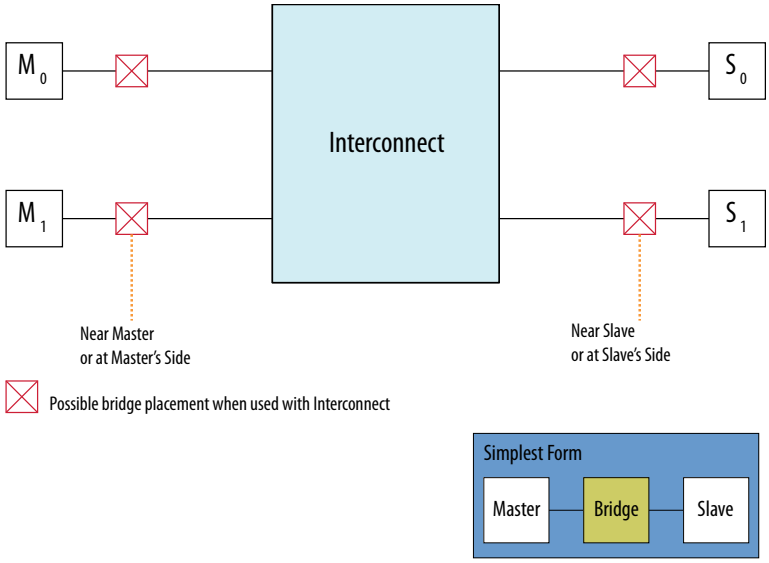


For a domain with multiple masters and slaves, placement of an AXI Timeout Bridge in your design may be beneficial in the following scenarios:

- To recover from a hang, place the bridge near the slave. If the master attempts to communicate with a slave that hangs, the AXI Timeout Bridge frees the master by generating error responses. The master is then able to communicate with another slave.
- When debugging your system, place the AXI Timeout Bridge near the master. This placement enables you to identify the origin of the burst and to obtain the full address from the master. Additionally, placing an AXI Timeout Bridge near the master enables you to identify the target slave for the burst.

  Note: If you put the bridge at the slave's side and you have multiple slaves connected to the same master, you do not get the full address.

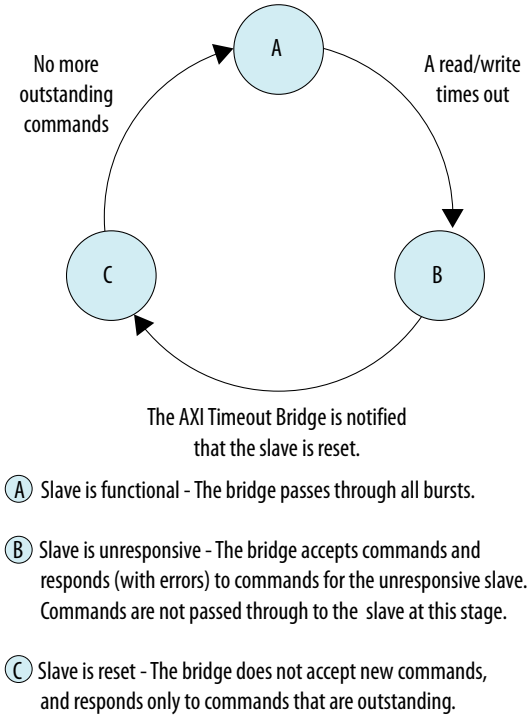
Figure 257. AXI Timeout Bridge Placement



14.1.7.1 AXI Timeout Bridge Stages

A timeout occurs when the internal timer in the bridge exceeds the specified number of cycles within which a burst must complete from start to end.

Figure 258. AXI Timeout Bridge Stages



- When a timeout occurs, the AXI Timeout Bridge asserts an interrupt and reports the burst that caused the timeout to the Configuration and Status Register (CSR).
- The bridge then generates error responses back to the master on behalf of the unresponsive slave. This stage frees the master and certifies the unresponsive slave as dysfunctional.
- The AXI Timeout Bridge accepts subsequent write addresses, write data, and read addresses to the dysfunctional slave. The bridge does not accept outstanding write responses, and read data from the dysfunctional slave is not passed through to the master.
- The `awvalid`, `wvalid`, `bready`, `arvalid`, and `rready` ports are held low at the master interface of the bridge.

Note: After a timeout, `awvalid`, `wvalid`, and `arvalid` may be dropped before they are accepted by `awready` at the master interface. While the behavior violates the AXI specification, it occurs only on an interface connected to the slave which has been certified dysfunctional by the AXI Timeout Bridge.

Write channel refers to the AXI write address, data and response channels. Similarly, read channel refers to the AXI read address and data channels. AXI write and read channels are independent of each other. However, when a timeout occurs on either channel, the bridge generates error responses on both channels.

Table 184. Burst Start and End Definitions for the AXI Timeout Bridge

Channel	Start	End
Write	When an address is issued. First cycle of `awvalid`, even if data of the same burst is issued before the address (first cycle of `wvalid`).	When the response is issued. First cycle of `bvalid`.
Read	When an address is issued. First cycle of `arvalid`.	When the last data is issued. First cycle of `rvalid` and `rlast`.

The AXI Timeout Bridge has four required interfaces: Master, Slave, Configuration and Status Register (CSR) (AXI4-Lite), and Interrupt. Qsys Pro allows the AXI Timeout Bridge to connect to any AXI3, AXI4, or Avalon master or slave interface. Avalon masters must utilize the bridge's interrupt output to detect a timeout.

The bridge slave interface accepts write addresses, write data, and read addresses, and then generates the SLVERR response at the write response and read data channels. You should not expect to use `buser`, `rdata` and `ruser` at this stage of processing.

To resume normal operation, the dysfunctional slave must be reset and the bridge notified of the change in status via the CSR. Once the CSR notifies the bridge that the slave is ready, the bridge does not accept new commands until all outstanding bursts are responded to with an error response.

The CSR has a 4-bit address width and a 32-bit data width. The CSR reports status and address information when the bridge asserts an interrupt.

Table 185. CSR Interrupt Status Information for the AXI Timeout Bridge

Address	Attribute	Name	Description
0x0	write-only	Slave is reset	Write a 1 to notify the AXI Timeout Bridge that the slave is reset and ready. Clears the interrupt.
0x4	read-only	Timed out operation	The operation of the burst that caused the timeout. 1 for a write; 0 for a read.
0x8 through 0xf	read-only	Timed out address	The address of the burst that caused the timeout. For an address width greater than 32-bits, CSR reads addresses 0x8 and 0xc to obtain the complete address.

14.1.7.2 AXI Timeout Bridge Parameters

Table 186. AXI Timeout Bridge Parameters

Parameter	Description
ID width	The width of `awid`, `bid`, `arid`, or `rid`.
Address width	The width of `awaddr` or `araddr`.
Data width	The width of `wdata` or `rdata`.
User width	The width of `awuser`, `wuser`, `buser`, `aruser`, or `ruser`.

continued...

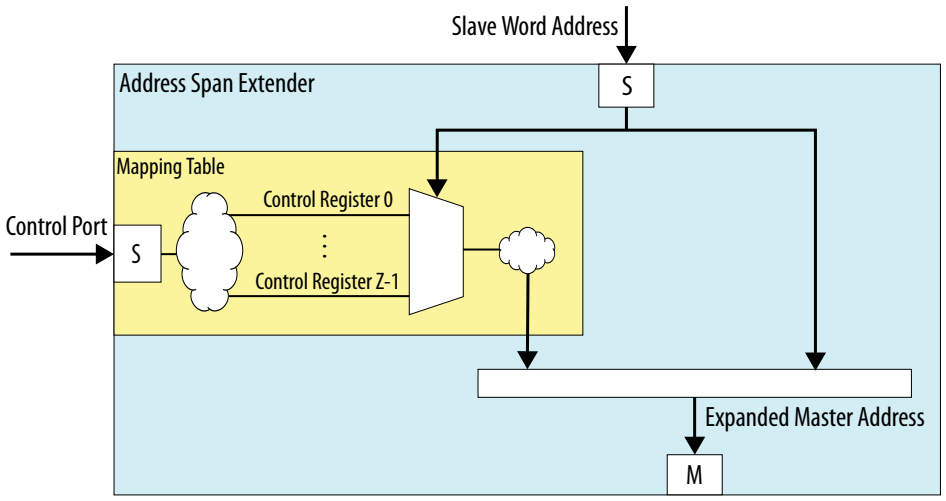
Parameter	Description
Maximum number of outstanding writes	The expected maximum number of outstanding writes.
Maximum number of outstanding reads	The expected maximum number of outstanding reads.
Maximum number of cycles	The number of cycles within which a burst must complete.

14.1.8 Address Span Extender

The **Address Span Extender** allows memory-mapped master interfaces to access a larger or smaller address map than the width of their address signals allows. The address span extender splits the addressable space into multiple separate windows, so that the master can access the appropriate part of the memory through the window.

The address span extender does not limit master and slave widths to a 32-bit and 64-bit configuration. You can use the address span extender with 1-64 bit address windows.

Figure 259. Address Span Extender



If a processor can address only 2 GB of an address span, and your system contains 4 GB of memory, the address span extender can provide two, 2 GB windows in the 4 GB memory address space. This issue sometimes occurs with Intel SoC devices.

For example, an HPS subsystem in an SoC device can address only 1 GB of an address span within the FPGA, using the HPS-to-FPGA bridge. The address span extender enables the SoC device to address all of the address space in the FPGA using multiple 1 GB windows.

Related Links

Qsys Pro 64-Bit Addressing Support on page 370
Qsys Pro interconnect supports up to 64-bit addressing for all Qsys Pro interfaces and IP components, with a range of: `0x0000 0000 0000 0000` to `0xFFFF FFFF FFFF FFFF`, inclusive.

14.1.8.1 CTRL Register Layout

The control registers consist of one 64-bit register for each window, where you specify the window's base address. For example, if CTRL_BASE is the base address of the control register, and address span extender contains two windows (0 and 1), then window 0's control register starts at CTRL_BASE, and window 1's control register starts at CTRL_BASE + 8 (using byte addresses).

14.1.8.2 Address Span Extender Parameters

Table 187. Address Span Extender Parameters

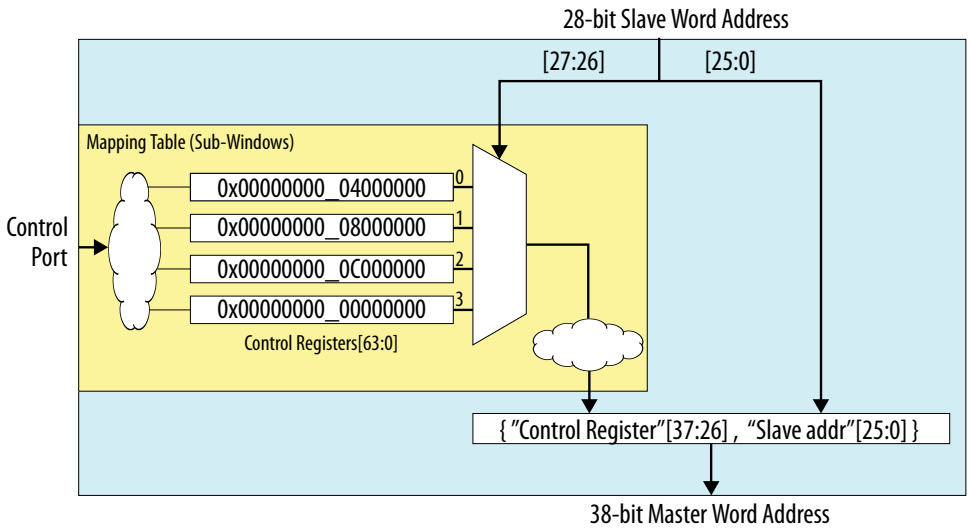
Parameter	Description
Datapath Width	Width of write data and read data signals.
Expanded Master Byte Address Width	Width of the master byte address port. That is, the address span size of all the downstream slaves that attach to the address span extender.
Slave Word Address Width	Width of the slave word address port. That is, the address span size of the downstream slaves that the upstream master accesses.
Burstcount Width	Burst count port width of the downstream slave and the upstream master that attach to the address span extender.
Number of sub-windows	The slave port can represent one or more windows in the master address span. You can subdivide the slave address span into *N* equal spans in *N* sub-windows. A remapping register in the CSR slave represents each sub-window, and configures the base address that each sub-window remaps to. The address span extender replaces the upper bits of the address with the stored index value in the remapping register before the master initiates a transaction.
Enable Slave Control Port	Dictates run-time control over the sub-window indexes. If you can define static re-mappings that do not need any change, you do not need to enable this CSR slave.
Maximum Pending Reads	Sets the bridge slave's `maximumPendingReadTransactions` property. In certain system configurations, you must increase this value to improve performance. This value usually aligns with the properties of the downstream slaves that you attach to this bridge.

14.1.8.3 Calculating the Address Span Extender Slave Address

The diagram describes how Qsys Pro calculates the slave address. In this example the address span extender is configured with a 28-bit address space for slaves. The upper 2 bits `[27:26]` are used to select the control registers.

The lower 26 bits (`[25:0]`) originate from the address span extender's data port, and are the offset into a particular window.

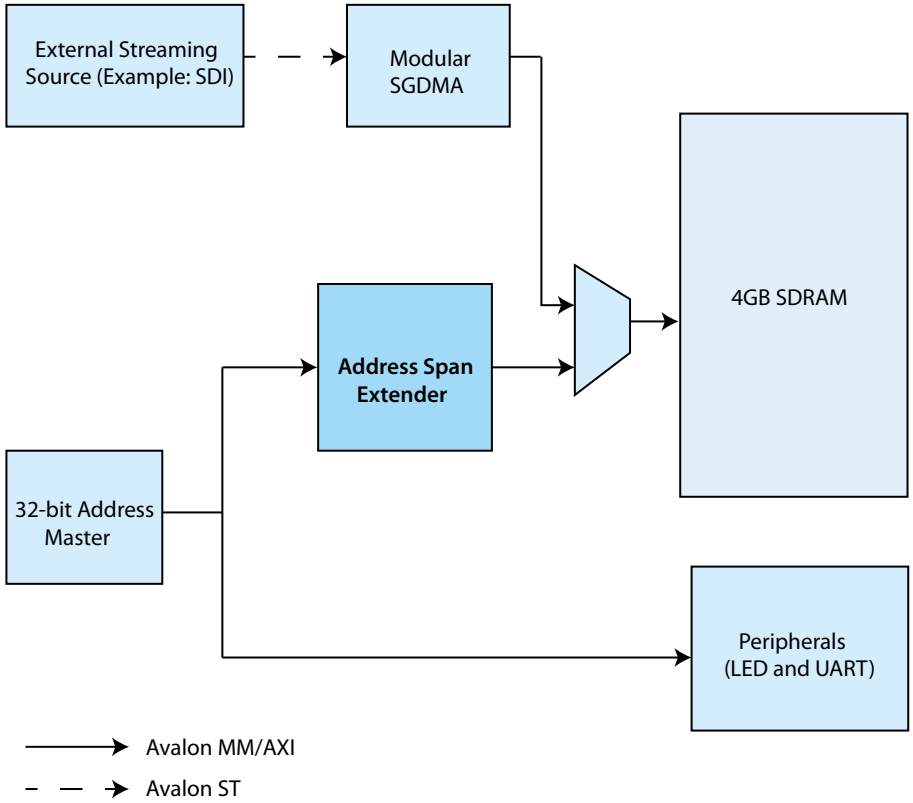
Figure 260. Address Span Extender



14.1.8.4 Using the Address Span Extender

This example shows when and how to use address span extender component in your Qsys Pro design.

Figure 261. Block Diagram with Address Span Extender



In the above design, a 32-bit master shares 4 GB SDRAM with an external streaming interface. The master has the path to access streaming data from the SDRAM DDR memory. However, if you connect the whole 32-bit address bus of the master to the SDRAM DDR memory, you cannot connect the master to peripherals such as LED or UART. To avoid this situation, you can implement the address span extender between the master and DDR memory. The address span extender allows the master to access the SDRAM DDR memory and the peripherals at the same time.

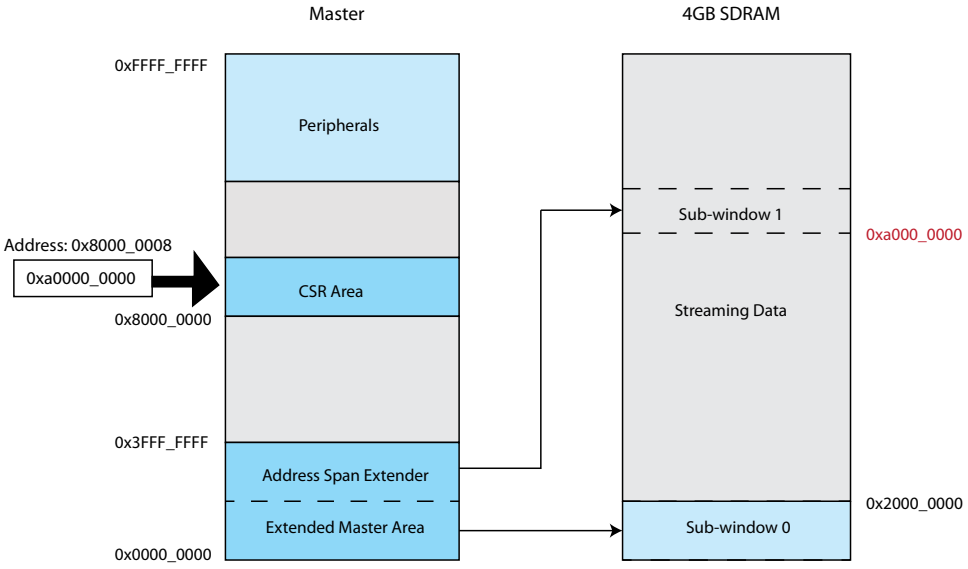
To implement address span extender for the above example, you can divide the address window of the address span extender into two sub-windows of 512 MB each. The sub-window 0 is for the master program area. You can dynamically map the sub-window 1 to any area other than the program area.

You can change the offset of the address window by setting the base address of sub-window 1 to the control register of the address span extender. However, you must make sure that the sub-window address span masks the base address. You can choose any arbitrary base address. If you set the value 0xa000_0000 to the control register, Qsys Pro maps the sub-window 1 to 0xa000_0000.

Table 188. CSR Mapping Table

Address	Data
0x8000_0000	0x0000_0000
0x8000_0008	0xa000_0000

Figure 262. Memory mapping for Address Span Extender

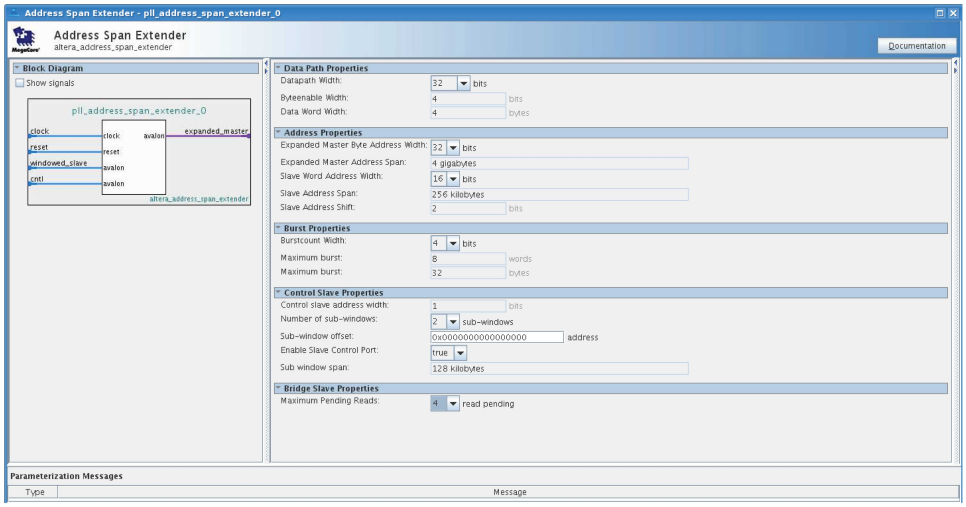


The table below indicates the Qsys Pro parameter settings for this address span extender example.

Table 189. Parameter Settings for the Address Span Extender Example

Parameter	Value	Description
Datapath Width	32 bits	The CPU has 32-bits data width and the SDRAM DDR memory has 512-bits data width. Since the transaction between the master and SDRAM DDR memory is minimal, set the datapath width to align with the upstream master.
Expanded Master Byte Address	32 bits	The address span extender has a 4 GB address span.
Slave Word Address Width	18 bits	There are two 512 MB sub-windows in reserve for the master. The number of bytes over the data word width in the **Datapath Properties** (4 bytes for this example) accounts for the slave address.
Burstcount Width	4 bits	The address span extender must handle up to 8 words burst in this example.
Number of sub-windows	2	Address window of the address span extender has two sub-windows of 512 MB each.
Enable Slave Control Port	true	The address span extender component must have control to change the base address of the sub-window.
Maximum Pending Reads	4	This number is the same as SDRAM DDR memory burst count.

Figure 263. Address Span Extender Parameter Editor



Note: You can view the address span extender connections in the **System Contents** tab. The windowed slave port and control port connect to the master. The expanded master port connects to the SDRAM DDR memory.

14.1.8.5 Alternate Options for the Address Span Extender

You can set parameters for the address span extender with an initial fixed address value. Enter an address for the **Reset Default for Master Window** option, and select **True** for the **Disable Slave Control Port** option. This allows the address span extender to function as a fixed, non-programmable component.

Each sub-window is equal in size and stacks sequentially in the windowed slave interface's address space. To control the fixed address bits of a particular sub-window, you can write to the sub-window's register in the register control slave interface. Qsys Pro structures the logic so that Qsys Pro can optimize and remove bits that are not needed.

If **Burstcount Width** is greater than 1, Qsys Pro processes the read burst in a single cycle, and assumes all `byteenables` are asserted on every cycle.

14.1.8.6 Nios II Support

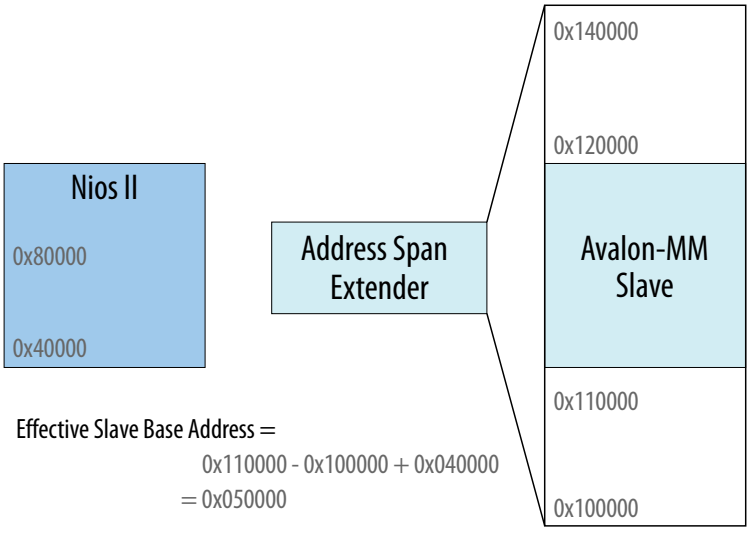
If the address span extender window is fixed, for example, the **Disable Slave Control Port** option is turned on, then the address span extender performs as a bridge. Components on the slave side of the address span extender that are within the window are visible to the Nios II processor. Components partially within a window appear to the Nios II processor as if they have a reduced span. For example, a memory partially within a window appears as having a smaller size.

You can also use the address span extender to provide a window for the Nios II processor so that the HPS memory map is visible to the Nios II processor. In this way it is possible for the Nios II processor to communicate with HPS peripherals.

In the example below, a Nios II processor has an address span extender from address `0x40000` to `0x80000`. There is a window within the address span extender starting at `0x100000`. Within the address span extender's address space there is a slave at base address `0x1100000`. The slave appears to the Nios II processor as being at address:

```
0x110000 - 0x100000 + 0x40000 = 0x050000
```

Figure 264. Nios II Support and the Address Span Extender



The address span extender window is dynamic. For example, when the **Disable Slave Control Port** option is turned off, the Nios II processor is unable to see components on the slave side of the address span extender.

14.2 AXI Default Slave

An AXI Default Slave provides a predictable error response service for master interfaces that send transactions that attempt to access an undefined memory region. This service guarantees an error response, should a master access a memory region that is not decoded to an instantiated slave. The error response service also helps to avoid unpredictable behavior in your system.

The default slave is an AXI3 component and displays in the IP Catalog as either **AXI Default Slave** or **Error Response Slave**.

AXI protocol requires that if the interconnect cannot successfully decode slave access, it must return the DECERR error response. Therefore, the default slave is required in AXI systems where the address space is not fully decoded to slave interfaces.

The default slave behaves like any other component in the system and is bound by translation and adaptation interconnect logic. An increase in resource usage may occur when a default slave connects to masters of different data widths, including Avalon or AXI-Lite masters.

You can connect clock, reset, and IRQ signals to a default slave, as well as AXI3 and AXI4 master interfaces without also instantiating a bridge. When you connect a default slave to a master, the default slave accepts cycles sent from the master, and returns the DECERR error response. On the AXI interface, the default slave supports only a read and write acceptance of 1, and does not support write data interleaving. The read and write channels are independent, and responses are returned when simultaneously targeted by a read and write cycle.

There is an optional interface on the default slave that supports CSR accesses for debug. CSR registers log the required information when returning an error response. When turned on, this channel acts as an Avalon interface with read and write channels with a fixed latency of 1.

To enable a slave interface as a default slave for a master interface in your system, you must connect the slave to the master in your Qsys Pro system. You specify a default slave for a master it by turning on the **Default Slave** column option in the **System Contents** tab. A system can contain more than one default slave. Intel recommends instantiating a separate default slave for each AXI master in your system.

Related Links

Creating a System with Qsys Pro on page 299
Qsys Pro is a system integration tool included as part of the Quartus Prime software.

14.2.1 AXI Default Slave Parameters

Figure 265. AXI Default Slave Parameter Editor

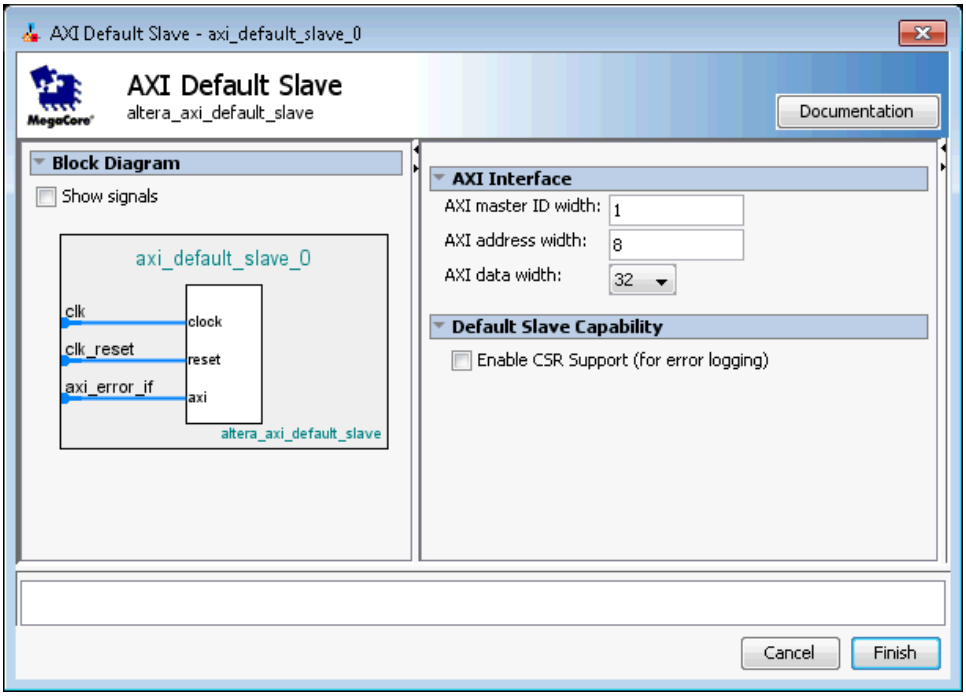


Table 190. AXI Default Slave Parameters

Parameter	Value	Description
AXI master ID width	1-8 bits	Determines the master ID width for error logging.
AXI address width	8-64 bits	Determines the address width for error logging. This value also affects the overall address width of the system, and should not exceed the maximum address width required in the system.
AXI data width	32, 64, or128 bits	Determines the data width for error logging.
Enable CSR Support (for error logging)	On or Off	When turned on, instantiates an Avalon CSR interface for error logging.
CSR Error Log Depth	1-16 bits	Depth of the transaction log, for example, the number of transactions the CSR logs for cycles with errors.
Register Avalon CSR inputs	On or Off	When turned on, controls debug access to the CSR interface.

14.2.2 CSR Registers

When an access violation occurs, and the CSR port is enabled, the AXI Default Slave generates an interrupt and transfers the transaction information into the error log FIFO.

The error log count continues until the n^{th} log, where *n* is the log depth. When Qsys Pro responds to the interrupt bit, it reads the register until the interrupt bit is no longer valid. The interrupt bit is valid as long as there is a valid bit in FIFO. A cleared interrupt bit is not affected by the FIFO status. When Qsys Pro finishes reading the register, the access violation service is ready to receive new access violation requests. If an access violation occurs when FIFO is full, then an overflow bit is set, indicating more than *n* access violations have occurred, and some are not logged.

Qsys Pro exits the access violation service after either the interrupt bit is no longer set, or when it determines that the access violation service has continued for too long.

14.2.2.1 CSR Interrupt Status Registers

Table 191. CSR Interrupt Status Registers

For CSR register maps: `Address = Memory Address Base + Offset`.

Offset	Bits	Attribute	Default	Descripton
0x00	31:4	RO	0	Reserved.
	3	RW1C	0	**Read Access Violation Interrupt Overflow register** Asserted when a read access causes the Interconnect to return a DECERR response, and the buffer log depth is full. Indicates that there is a logging error lost due to an exceeded buffer log depth. Cleared by setting the bit to 1.
	2	RW1C	0	**Write Access Violation Interrupt Overflow register** Asserted when a write access causes the Interconnect to return a DECERR response, and the buffer log depth is full. Indicates that there is a logging error lost due to an exceeded buffer log depth. Cleared by setting the bit to 1.
	1	RW1C	0	**Read Access Violation Interrupt register** Asserted when a read access causes the Interconnect to return a DECERR response. Cleared by setting the bit to 1. *Note:* Access violation are logged until the bit is cleared.
	0	RW1C	0	**Write Access Violation Interrupt register** Asserted when a write access causes the Interconnect to return a DECERR response. Cleared by setting the bit to 1. *Note:* Access violation are logged until the bit is cleared.

14.2.2.2 CSR Read Access Violation Log

The CSR read access violation log settings are valid only when an associated read interrupt register is set. This set of registers should be read until the valid bit is cleared.

Table 192. CSR Read Access Violation Log

Offset	Bits	Attribute	Default	Description
0x100	31:13	RO	0	Reserved.
	12:11	RO	0	Indicates the burst type of the initiating cycle that causes the access violation.
	10:7	RO	0	Indicates the burst length of the initiating cycle that causes the access violation.
	6:4	RO	0	Indicates the burst size of the initiating cycle that causes the access violation.

continued...

Offset	Bits	Attribute	Default	Description
	3:1	R0	0	Indicates the PROT of the initiating cycle that causes the access violation.
	0	R0	0	Read access violation log for the transaction is valid only when this bit is set. This bit is cleared when the interrupt register is cleared.
0x104	31:0	R0	0	Master ID for the cycle that causes the access violation.
0x108	31:0	R0	0	Read cycle target address for the cycle that causes the access violation (lower 32-bit).
0x10C	31:0	R0	0	Read cycle target address for the cycle that causes the access violation (upper 32-bit). Valid only if widest address in system is larger than 32 bits. *Note:* When this register is read, the current read access violation log is recovered from FIFO.

14.2.2.3 CSR Write Access Violation Log

The CSR write access violation log settings are valid only when an associated read interrupt register is set. This set of registers should be read until the valid bit is cleared.

Table 193. CSR Write Access Violation Log

Offset	Bits	Attribute	Default	Description
0x190	31:13	R0	0	Reserved.
	12:11	R0	0	Indicates the burst type of the initiating cycle that causes the access violation.
	10:7	R0	0	Indicates the burst length of the initiating cycle that causes the access violation.
	6:4	R0	0	Indicates the burst size of the initiating cycle that causes the access violation.
	3:1	R0	0	Indicates the PROT of the initiating cycle that causes the access violation.
	0	R0	0	Write access violation log for the transaction is valid only when this bit is set. This bit is cleared when the interrupt register is cleared.
0x194	31:0	R0	0	Master ID for the cycle that causes the access violation.
0x198	31:0	R0	0	Write target address for the cycle that causes the access violation (lower 32-bit).
0x19C	31:0	R0	0	Write target address for the cycle that causes the access violation (upper 32-bit). Valid only if widest address in system is larger than 32 bits.
0x1A0	31:0	R0	0	First 32 bits of the write data for the write cycle that causes the access violation. *Note:* When this register is read, the current write access violation log is recovered from FIFO, when the data width is 32 bits.

continued...

Offset	Bits	Attribute	Default	Description
0x1A4	31:0	RO	0	Bits [63:32] of the write data for the write cycle that causes the access violation. Valid only if the data width is greater than 32 bits.
0x1A8	31:0	RO	0	Bits [95:64] of the write data for the write cycle that causes the access violation. Valid only if the data width is greater than 64 bits.
0x1AC	31:0	RO	0	The first bits [127:96] of the write data for the write cycle that causes the access violation. Valid only if the data width is greater than 64 bits. *Note:* When this register is read, the current write access violation log is recovered from FIFO.

14.2.3 Designating a Default Slave in the System Contents Tab

You can designate any slave in your Qsys Pro system as the error response default slave. The designated default slave provides an error response service for masters that attempt access to an undefined memory region.

1. In your Qsys Pro system, in the **System Contents** tab, right-click the header and turn on **Show Default Slave Column**.
2. Select the slave that you want to designate as the default slave, and then click the checkbox for the slave in the **Default Slave** column.
3. In the **System Contents** tab, in the **Connections** column, connect the designated default slave to one or more masters.

14.3 Tri-State Components

The tri-state interface type allows you to design Qsys Pro subsystems that connect to tri-state devices on your PCB. You can use tri-state components to implement pin sharing, convert between unidirectional and bidirectional signals, and create tri-state controllers for devices whose interfaces can be described using the tri-state signal types.

Figure 266. Tri-State Conduit System to Control Off-Chip SRAM and Flash Devices

In this example, there are two generic Tri-State Conduit Controllers. The first is customized to control a flash memory. The second is customized to control an off-chip SSRAM. The Tri-State Conduit Pin Sharer multiplexes between these two controllers, and the Tri-State Conduit Bridge converts between an on-chip encoding of tri-state signals and true bidirectional signals. By default, the Tri-State Conduit Pin Sharer and Tri-State Conduit Bridge present byte addresses. Typically, each address location contains more than one byte of data.

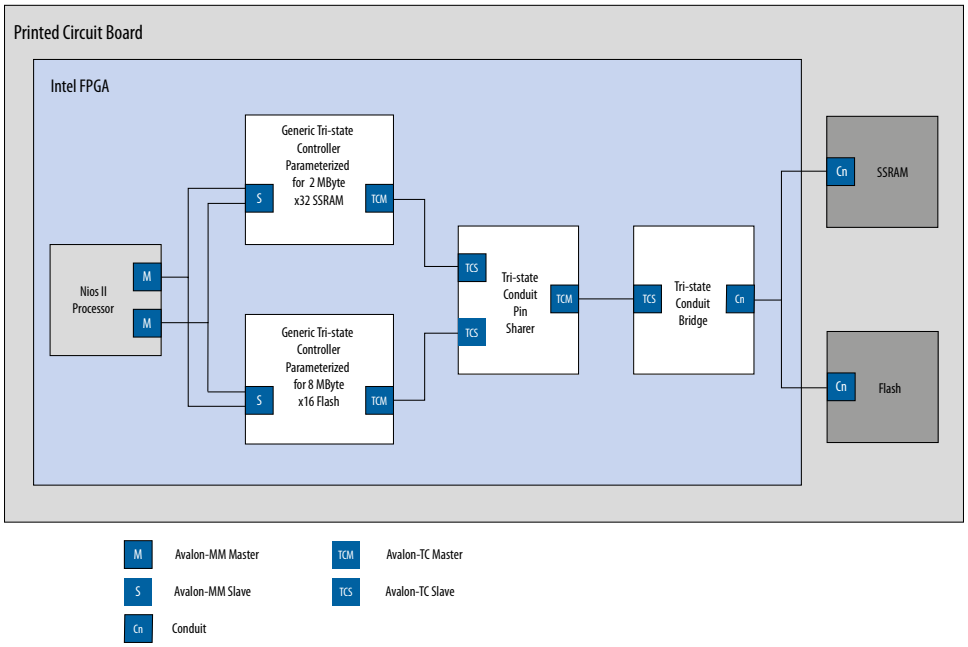
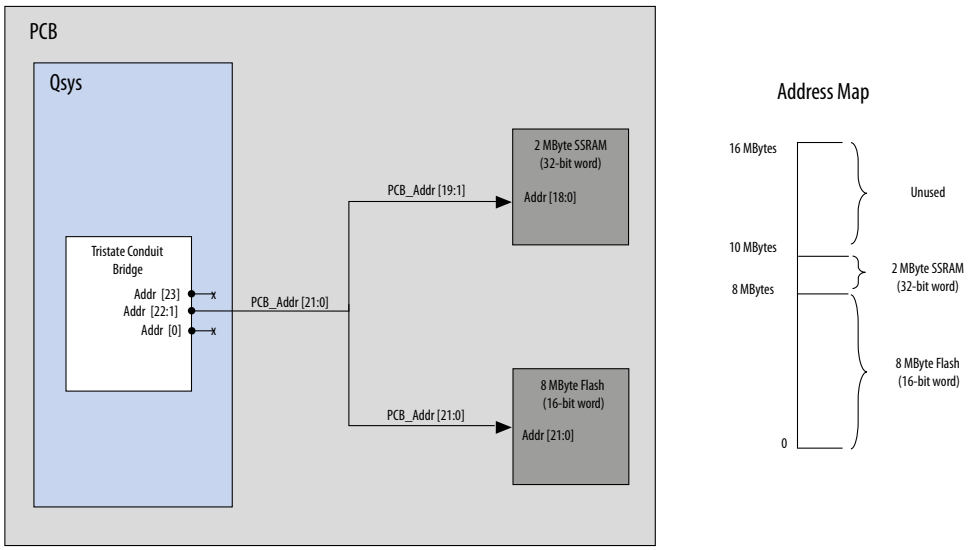


Figure 267. Address Connections from Qsys Pro System to PCB

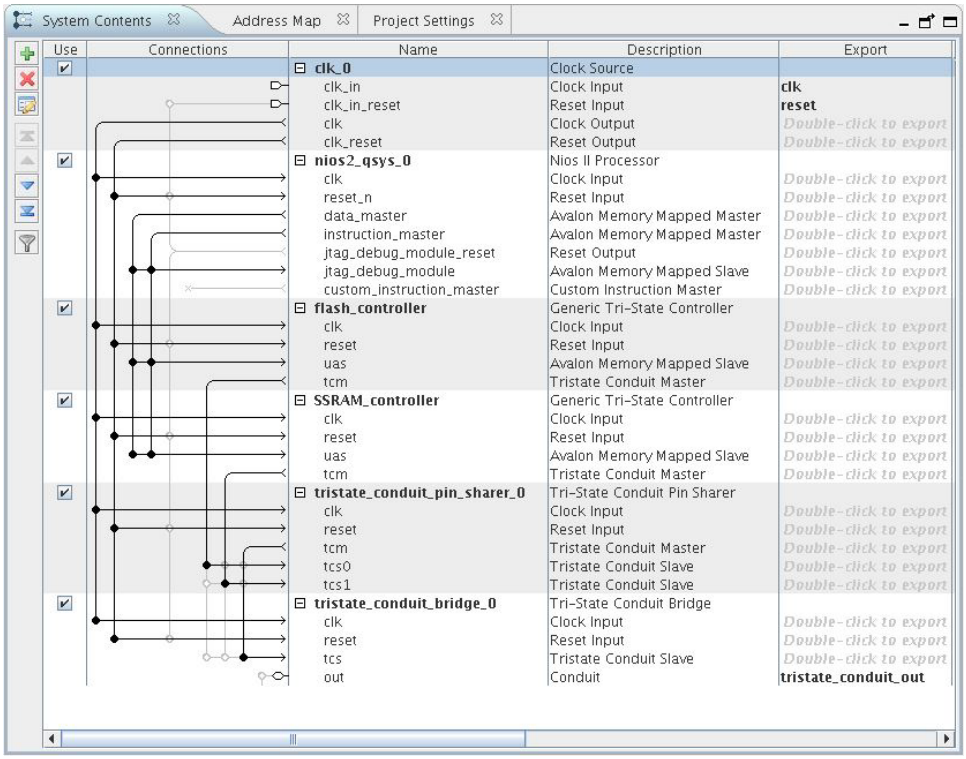
The flash device operates on 16-bit words and must ignore the least-significant bit of the Avalon-MM address. The figure shows `addr[0]`as not connected. The SSRAM memory operates on 32-bit words and must ignore the two low-order memory bits. Because neither device requires a byte address, `addr[0]` is not routed on the PCB.

The flash device responds to address range 0 MBytes to 8 MBytes-1. The SSRAM responds to address range 8 MBytes to 10 MBytes-1. The PCB schematic for the PCB connects `addr[21:0]` to `addr[18:0]` of the SSRAM device because the SSRAM responds to 32-bit word address. The 8 MByte flash device accesses 16-bit words; consequently, the schematic does not connect `addr[0]`. The `chipselect` signals select between the two devices.



Note: If you create a custom tri-state conduit master with word aligned addresses, the Tri-state Conduit Pin Sharer does not change or align the address signals.

Figure 268. Tri-State Conduit System in Qsys Pro



Related Links

- Avalon Interface Specifications
- Avalon Tri-State Conduit Components User Guide

14.3.1 Generic Tri-State Controller

The Generic Tri-State Controller provides a template for a controller. You can customize the tri-state controller with various parameters to reflect the behavior of an off-chip device. The following types of parameters are available for the tri-state controller:

- Width of the address and data signals
- Read and write wait times
- Bus-turnaround time
- Data hold time

Note: In calculating delays, the Generic Tri-State Controller chooses the larger of the bus-turnaround time and read latency. Turnaround time is measured from the time that a command is accepted, not from the time that the previous read returned data.

The Generic Tri-State Controller includes the following interfaces:

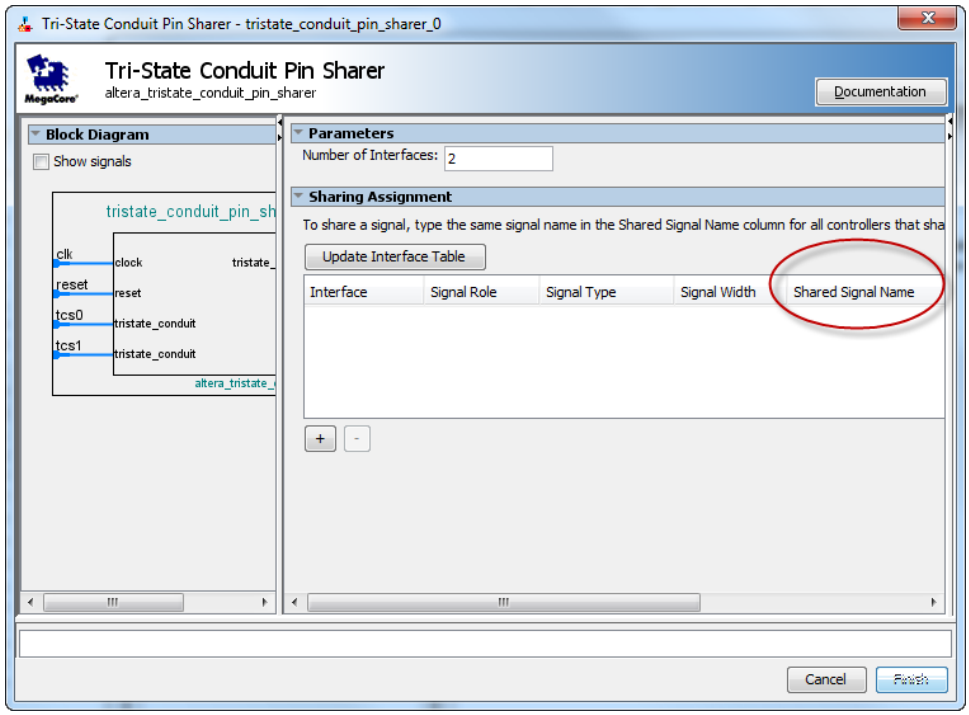
- **Memory-mapped slave interface**—This interface connects to a memory-mapped master, such as a processor.
- **Tristate Conduit Master interface**—The tri-state master interface usually connects to the tri-state conduit slave interface of the tri-state conduit pin sharer.
- **Clock sink**—The component's clock reference. You must connect this interface to a clock source.
- **Reset sink**—This interface connects to a reset source interface.

14.3.2 Tri-State Conduit Pin Sharer

The Tri-state Conduit Pin Sharer multiplexes between the signals of the connected tri-state controllers. You connect all signals from the tri-state controllers to the Tri-state Conduit Pin Sharer and use the parameter editor to specify the signals that are shared.

Figure 269. Tri-State Conduit Pin Sharer Parameter Editor

The parameter editor includes a **Shared Signal Name** column. If the widths of shared signals differ, the signals are aligned on their 0^{th} bit and the higher-order pins are driven to 0 whenever the smaller signal has control of the bus. Unshared signals always propagate through the pin sharer. The tri-state conduit pin sharer uses the round-robin arbiter to select between tri-state conduit controllers.



Note: All tri-state conduit components connected to a pin sharer must be in the same clock domain.

Related Links

Avalon-ST Round Robin Scheduler on page 936

14.3.3 Tri-State Conduit Bridge

The Tri-State Conduit Bridge instantiates bidirectional signals for each tri-state signal while passing all other signals straight through the component. The Tri-State Conduit Bridge registers all outgoing and incoming signals, which adds two cycles of latency for a read request. You must account for this additional pipelining when designing a custom controller. During reset, all outputs are placed in a high-impedance state. Outputs are enabled in the first clock cycle after reset is deasserted, and the output signals are then bidirectional.

14.4 Test Pattern Generator and Checker Cores

The test pattern generator inserts different error conditions, and the test pattern checker reports these error conditions to the control interface, each via an Avalon Memory-Mapped (Avalon-MM) slave.

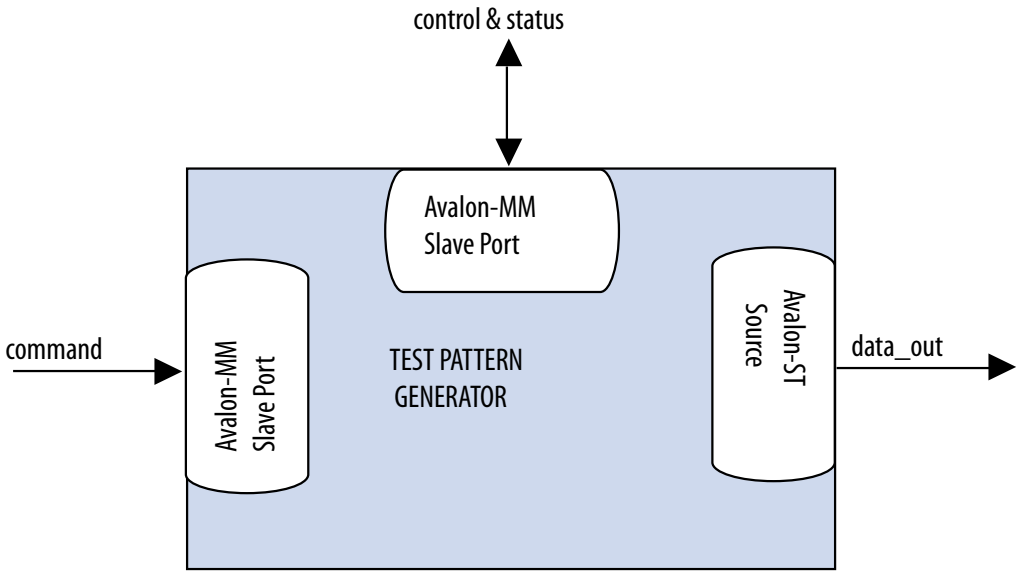
The data generation and monitoring solution for Avalon-ST consists of two components: a test pattern generator core that generates data, and sends it out on an Avalon-ST data interface, and a test pattern checker core that receives the same data and verifies it. Optionally, the data can be formatted as packets, with accompanying `start_of_packet` and `end_of_packet` signals.

The **Throttle Seed** is the starting value for the throttle control random number generator. Intel recommends a unique value for each instance of the test pattern generator and checker cores in a system.

14.4.1 Test Pattern Generator

Figure 270. Test Pattern Generator Core

The test pattern generator core accepts commands to generate data via an Avalon-MM command interface, and drives the generated data to an Avalon-ST data interface. You can parameterize most aspects of the Avalon-ST data interface, such as the number of error bits and data signal width, thus allowing you to test components with different interfaces.



The data pattern is calculated as: *Symbol Value* = *Symbol Position in Packet* XOR *Data Error Mask*. Data that is not organized in packets is a single stream with no beginning or end. The test pattern generator has a throttle register that is set via the Avalon-MM control interface. The test pattern generator uses the value of the throttle register in conjunction with a pseudo-random number generator to throttle the data generation rate.

14.4.1.1 Test Pattern Generator Command Interface

The command interface for the Test Pattern Generator is a 32-bit Avalon-MM write slave that accepts data generation commands. It is connected to a 16-element deep FIFO, thus allowing a master peripheral to drive a number of commands into the test pattern generator.

The command interface maps to the following registers: `cmd_lo` and `cmd_hi`. The command is pushed into the FIFO when the register `cmd_lo` (address 0) is addressed. When the FIFO is full, the command interface asserts the `waitrequest` signal. You can create errors by writing to the register `cmd_hi` (address 1). The errors are cleared when 0 is written to this register, or its respective fields.

14.4.1.2 Test Pattern Generator Control and Status Interface

The control and status interface of the Test Pattern Generator is a 32-bit Avalon-MM slave that allows you to enable or disable the data generation, as well as set the throttle. This interface also provides generation-time information, such as the number of channels and whether or not data packets are supported.

14.4.1.3 Test Pattern Generator Output Interface

The output interface of the Test Pattern Generator is an Avalon-ST interface that optionally supports data packets. You can configure the output interface to align with your system requirements. Depending on the incoming stream of commands, the output data may contain interleaved packet fragments for different channels. To keep track of the current symbol's position within each packet, the test pattern generator maintains an internal state for each channel.

You can configure the output interface of the test pattern generator with the following parameters:

- **Number of Channels**—Number of channels that the test pattern generator supports. Valid values are 1 to 256.
- **Data Bits Per Symbol**—Bits per symbol is related to the width of `readdata` and `writedata` signals, which must be a multiple of the bits per symbol.
- **Data Symbols Per Beat**—Number of symbols (words) that are transferred per beat. Valid values are 1 to 256.
- **Include Packet Support**—Indicates whether or not packet transfers are supported. Packet support includes the `startofpacket`, `endofpacket`, and `empty` signals.
- **Error Signal Width (bits)**—Width of the error signal on the output interface. Valid values are 0 to 31. A value of `0` indicates that the error signal is not in use.

Note: If you change only bits per symbol, and do not change the data width, errors are generated.

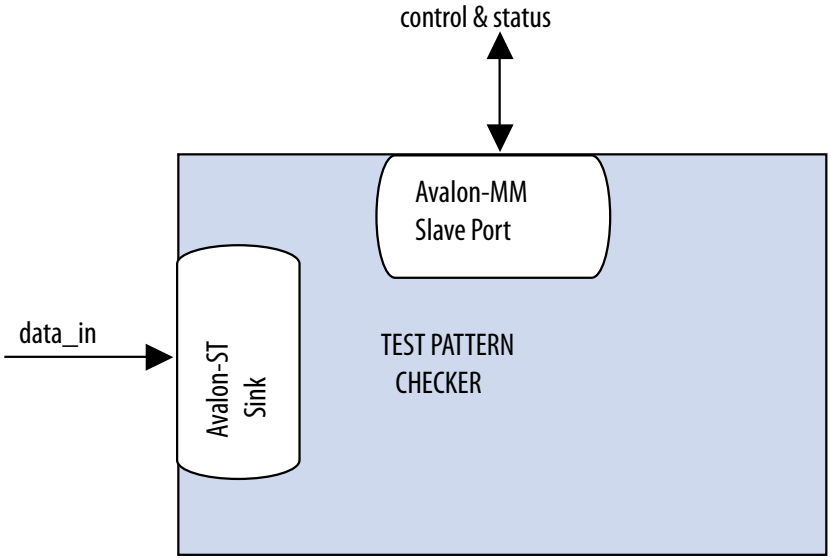
14.4.1.4 Test Pattern Generator Functional Parameter

The Test Pattern Generator functional parameter allows you to configure the test pattern generator as a whole system.

14.4.2 Test Pattern Checker

Figure 271. Test Pattern Checker

The test pattern checker core accepts data via an Avalon-ST interface and verifies it against the same predetermined pattern that the test pattern generator uses to produce the data. The test pattern checker core reports any exceptions to the control interface. You can parameterize most aspects of the test pattern checker's Avalon-ST interface such as the number of error bits and the data signal width. This enables the ability to test components with different interfaces. The test pattern checker has a throttle register that is set via the Avalon-MM control interface. The value of the throttle register controls the rate at which data is accepted.



The test pattern checker detects exceptions and reports them to the control interface via a 32-element deep internal FIFO. Possible exceptions are data error, missing start-of-packet (SOP), missing end-of-packet (EOP), and signaled error.

As each exception occurs, an exception descriptor is pushed into the FIFO. If the same exception occurs more than once consecutively, only one exception descriptor is pushed into the FIFO. All exceptions are ignored when the FIFO is full. Exception descriptors are deleted from the FIFO after they are read by the control and status interface.

14.4.2.1 Test Pattern Checker Input Interface

The Test Pattern Checker input interface is an Avalon-ST interface that optionally supports data packets. You can configure the input interface to align with your system requirements. Incoming data may contain interleaved packet fragments. To keep track of the current symbol’s position, the test pattern checker maintains an internal state for each channel.

14.4.2.2 Test Pattern Checker Control and Status Interface

The Test Pattern Checker control and status interface is a 32-bit Avalon-MM slave that allows you to enable or disable data acceptance, as well as set the throttle. This interface provides generation-time information, such as the number of channels and

whether the test pattern checker supports data packets. The control and status interface also provides information on the exceptions detected by the test pattern checker. The interface obtains this information by reading from the exception FIFO.

14.4.2.3 Test Pattern Checker Functional Parameter

The Test Pattern Checker functional parameter allows you to configure the test pattern checker as a whole system.

14.4.2.4 Test Pattern Checker Input Parameters

You can configure the input interface of the test pattern checker using the following parameters:

- **Data Bits Per Symbol**—Bits per symbol is related to the width of `readdata` and `writedata` signals, which must be a multiple of the bits per symbol.
- **Data Symbols Per Beat**—Number of symbols (words) that are transferred per beat. Valid values are 1 to 32.
- **Include Packet Support**—Indicates whether or not data packet transfers are supported. Packet support includes the `startofpacket`, `endofpacket`, and `empty` signals.
- **Number of Channels**—Number of channels that the test pattern checker supports. Valid values are 1 to 256.
- **Error Signal Width (bits)**—Width of the `error` signal on the input interface. Valid values are 0 to 31. A value of `0` indicates that the `error` signal in not in use.

Note: If you change only bits per symbol, and do not change the data width, errors are generated.

14.4.3 Software Programming Model for the Test Pattern Generator and Checker Cores

The HAL system library support, software files, and register maps describe the software programming model for the test pattern generator and checker cores.

14.4.3.1 HAL System Library Support

For Nios II processor users, Intel provides HAL system library drivers that allow you to initialize and access the test pattern generator and checker cores. Intel recommends you use the provided drivers to access the cores instead of accessing the registers directly.

For Nios II IDE users, copy the provided drivers from the following installation folders to your software application directory:

- `<IP installation directory>/ip/sopc_builder_ip/altera_Avalon_data_source/HAL`
- `<IP installation directory>/ip/sopc_builder_ip/altera_Avalon_data_sink/HAL`

Note: This instruction does not apply if you use the Nios II command-line tools.

14.4.3.2 Test Pattern Generator and Test Pattern Checker Core Files

The following files define the low-level access to the hardware, and provide the routines for the HAL device drivers.

Note: Do not modify the test pattern generator or test pattern checker core files.

- Test pattern generator core files:
  - **data_source_regs.h**—Header file that defines the test pattern generator's register maps.
  - **data_source_util.h , data_source_util.c**—Header and source code for the functions and variables required to integrate the driver into the HAL system library.
- Test pattern checker core files:
  - **data_sink_regs.h**—Header file that defines the core's register maps.
  - **data_sink_util.h , data_sink_util.c**—Header and source code for the functions and variables required to integrate the driver into the HAL system library.

14.4.3.3 Register Maps for the Test Pattern Generator and Test Pattern Checker Cores

14.4.3.3.1 Test Pattern Generator Control and Status Registers

Table 194. Test Pattern Generator Control and Status Register Map

Shows the offset for the test pattern generator control and status registers. Each register is 32-bits wide.

Offset	Register Name
base + 0	`status`
base + 1	`control`
base + 2	`fill`

Table 195. Test Pattern Generator Status Register Bits

Bit(s)	Name	Access	Description
[15:0]	`ID`	RO	A constant value of `0x64`.
[23:16]	`NUMCHANNELS`	RO	The configured number of channels.
[30:24]	`NUMSYMBOLS`	RO	The configured number of symbols per beat.
[31]	`SUPPORTPACKETS`	RO	A value of 1 indicates data packet support.

Table 196. Test Pattern Generator Control Register Bits

Bit(s)	Name	Access	Description
[0]	`ENABLE`	RW	Setting this bit to 1 enables the test pattern generator core.
[7:1]	Reserved		

continued...

Bit(s)	Name	Access	Description
[16:8]	THROTTLE	RW	Specifies the throttle value which can be between 0–256, inclusively. The test pattern generator uses this value in conjunction with a pseudo-random number generator to throttle the data generation rate. Setting THROTTLE to 0 stops the test pattern generator core. Setting it to 256 causes the test pattern generator core to run at full throttle. Values between 0–256 result in a data rate proportional to the throttle value.
[17]	SOFT RESET	RW	When this bit is set to 1, all internal counters and statistics are reset. Write 0 to this bit to exit reset.
[31:18]	Reserved		

Table 197. Test Pattern Generator Fill Register Bits

Bit(s)	Name	Access	Description
[0]	BUSY	RO	A value of 1 indicates that data transmission is in progress, or that there is at least one command in the command queue.
[6:1]	Reserved		
[15:7]	FILL	RO	The number of commands currently in the command FIFO.
[31:16]	Reserved		

14.4.3.3.2 Test Pattern Generator Command Registers

Table 198. Test Pattern Generator Command Register Map

Shows the offset for the command registers. Each register is 32-bits wide.

Offset	Register Name
base + 0	cmd_lo
base + 1	cmd_hi

The cmd_lo is pushed into the FIFO only when the cmd_lo register is addressed.

Table 199. cmd_lo Register Bits

Bit(s)	Name	Access	Description
[15:0]	SIZE	RW	The segment size in symbols. Except for the last segment in a packet, the size of all segments must be a multiple of the configured number of symbols per beat. If this condition is not met, the test pattern generator core inserts additional symbols to the segment to ensure the condition is fulfilled.
[29:16]	CHANNEL	RW	The channel to send the segment on. If the channel signal is less than 14 bits wide, the test pattern generator uses the low order bits of this register to drive the signal.
[30]	SOP	RW	Set this bit to 1 when sending the first segment in a packet. This bit is ignored when data packets are not supported.
[31]	EOP	RW	Set this bit to 1 when sending the last segment in a packet. This bit is ignored when data packets are not supported.

Table 200. cmd_hi Register Bits

Bit(s)	Name	Access	Description
[15:0]	SIGNALED ERROR	RW	Specifies the value to drive the error signal. A non-zero value creates a signalled error.
[23:16]	DATA ERROR	RW	The output data is XORed with the contents of this register to create data errors. To stop creating data errors, set this register to 0.
[24]	SUPPRESS SOP	RW	Set this bit to 1 to suppress the assertion of the startofpacket signal when the first segment in a packet is sent.
[25]	SUPRESS EOP	RW	Set this bit to 1 to suppress the assertion of the endofpacket signal when the last segment in a packet is sent.

14.4.3.3.3 Test Pattern Checker Control and Status Registers

Table 201. Test Pattern Checker Control and Status Register Map

Shows the offset for the control and status registers. Each register is 32 bits wide.

Offset	Register Name
base + 0	status
base + 1	control
base + 2	Reserved
base + 3	
base + 4	
base + 5	exception_descriptor
base + 6	indirect_select
base + 7	indirect_count

Table 202. Test Pattern Checker Status Register Bits

Bit(s)	Name	Access	Description
[15:0]	ID	RO	Contains a constant value of 0x65.
[23:16]	NUMCHANNELS	RO	The configured number of channels.
[30:24]	NUMSYMBOLS	RO	The configured number of symbols per beat.
[31]	SUPPORTPACKETS	RO	A value of 1 indicates packet support.

Table 203. Test Pattern Checker Control Register Bits

Bit(s)	Name	Access	Description
[0]	ENABLE	RW	Setting this bit to 1 enables the test pattern checker.
[7:1]	Reserved		
[16:8]	THROTTLE	RW	Specifies the throttle value which can be between 0–256, inclusively. Qsys Pro uses this value in conjunction with a pseudo-random number generator to throttle the data generation rate.

continued...

Bit(s)	Name	Access	Description
			Setting THROTTLE to 0 stops the test pattern generator core. Setting it to 256 causes the test pattern generator core to run at full throttle. Values between 0–256 result in a data rate proportional to the throttle value.
[17]	SOFT RESET	RW	When this bit is set to 1, all internal counters and statistics are reset. Write 0 to this bit to exit reset.
[31:18]	Reserved		

If there is no exception, reading the `exception_descriptor` register bit register returns 0.

Table 204. `exception_descriptor` Register Bits

Bit(s)	Name	Access	Description
[0]	DATA ERROR	RO	A value of 1 indicates that an error is detected in the incoming data.
[1]	MISSINGSOP	RO	A value of 1 indicates missing start-of-packet.
[2]	MISSINGEOP	RO	A value of 1 indicates missing end-of-packet.
[7:3]	Reserved		
[15:8]	SIGNALLED ERROR	RO	The value of the `error` signal.
[23:16]	Reserved		
[31:24]	CHANNEL	RO	The channel on which the exception was detected.

Table 205. `indirect_select` Register Bits

Bit	Bits Name	Access	Description
[7:0]	INDIRECT CHANNEL	RW	Specifies the channel number that applies to the INDIRECT PACKET COUNT, INDIRECT SYMBOL COUNT, and INDIRECT ERROR COUNT registers.
[15:8]	Reserved		
[31:16]	INDIRECT ERROR	RO	The number of data errors that occurred on the channel specified by INDIRECT CHANNEL.

Table 206. `indirect_count` Register Bits

Bit	Bits Name	Access	Description
[15:0]	INDIRECT PACKET COUNT	RO	The number of data packets received on the channel specified by INDIRECT CHANNEL.
[31:16]	INDIRECT SYMBOL COUNT	RO	The number of symbols received on the channel specified by INDIRECT CHANNEL.

.

14.4.4 Test Pattern Generator API

The following subsections describe application programming interface (API) for the test pattern generator.

Note: API functions are currently not available from the interrupt service routine (ISR).

14.4.4.1 data_source_reset()

Table 207. data_source_reset()

Information Type	Description
Prototype	`void data_source_reset(alt_u32 base);`
Thread-safe	No
Include	*<data_source_util.h >*
Parameters	`base`—Base address of the control and status slave.
Returns	`void`
Description	Resets the test pattern generator core including all internal counters and FIFOs. The control and status registers are not reset by this function.

14.4.4.2 data_source_init()

Table 208. data_source_init()

Information Type	Description
Prototype	`int data_source_init(alt_u32 base, alt_u32 command_base);`
Thread-safe	No
Include	*<data_source_util.h >*
Parameters	`base`—Base address of the control and status slave. `command_base`—Base address of the command slave.
Returns	`1`—Initialization is successful.

continued...

Information Type	Description
	0—Initialization is unsuccessful.
Description	Performs the following operations to initialize the test pattern generator core: • Resets and disables the test pattern generator core. • Sets the maximum throttle. • Clears all inserted errors.

14.4.4.3 data_source_get_id()

Table 209. data_source_get_id()

Information Type	Description
Prototype	`int data_source_get_id(alt_u32 base);`
Thread-safe	Yes
Include	*<data_source_util.h >*
Parameters	base—Base address of the control and status slave.
Returns	Test pattern generator core identifier.
Description	Retrieves the test pattern generator core's identifier.

14.4.4.4 data_source_get_supports_packets()

Table 210. data_source_get_supports_packets()

Information Type	Description
Prototype	`int data_source_init(alt_u32 base);`
Thread-safe	Yes
Include	*<data_source_util.h >*
Parameters	base—Base address of the control and status slave.
Returns	1—Data packets are supported. 0—Data packets are not supported.
Description	Checks if the test pattern generator core supports data packets.

14.4.4.5 data_source_get_num_channels()

Table 211. data_source_get_num_channels()

Description	Description
Prototype	`int data_source_get_num_channels(alt_u32 base);`
Thread-safe	Yes
Include	*<data_source_util.h >*
Parameters	base—Base address of the control and status slave.
Returns	Number of channels supported.
Description	Retrieves the number of channels supported by the test pattern generator core.

14.4.4.6 data_source_get_symbols_per_cycle()

Table 212. data_source_get_symbols_per_cycle()

Description	Description
Prototype	`int data_source_get_symbols(alt_u32 base);`
Thread-safe	Yes
Include	*<data_source_util.h >*
Parameters	`base`—Base address of the control and status slave.
Returns	Number of symbols transferred in a beat.
Description	Retrieves the number of symbols transferred by the test pattern generator core in each beat.

14.4.4.7 data_source_get_enable()

Table 213. data_source_get_enable()

Information Type	Description
Prototype	`int data_source_get_enable(alt_u32 base);`
Thread-safe	Yes
Include	*<data_source_util.h >*
Parameters	`base`—Base address of the control and status slave.
Returns	Value of the ENABLE bit.
Description	Retrieves the value of the ENABLE bit.

14.4.4.8 data_source_set_enable()

Table 214. data_source_set_enable()

Information Type	Description
Prototype	`void data_source_set_enable(alt_u32 base, alt_u32 value);`
Thread-safe	No
Include	*<data_source_util.h >*
Parameters	`base`—Base address of the control and status slave. `value`— ENABLE bit set to the value of this parameter.
Returns	`void`
Description	Enables or disables the test pattern generator core. When disabled, the test pattern generator core stops data transmission but continues to accept commands and stores them in the FIFO

14.4.4.9 data_source_get_throttle()

Table 215. data_source_get_throttle()

Information Type	Description
Prototype	`int data_source_get_throttle(alt_u32 base);`
Thread-safe	Yes
Include	*<data_source_util.h >*
Parameters	`base`—Base address of the control and status slave.
Returns	Throttle value.
Description	Retrieves the current throttle value.

14.4.4.10 data_source_set_throttle()

Table 216. data_source_set_throttle()

Information Type	Description
Prototype	`void data_source_set_throttle(alt_u32 base, alt_u32 value);`
Thread-safe	No
Include	*<data_source_util.h >*
Parameters	`base`—Base address of the control and status slave. `value`—Throttle value.
Returns	`void`
Description	Sets the throttle value, which can be between 0–256 inclusively. The throttle value, when divided by 256 yields the rate at which the test pattern generator sends data.

14.4.4.11 data_source_is_busy()

Table 217. data_source_is_busy()

Information Type	Description
Prototype	`int data_source_is_busy(alt_u32 base);`
Thread-safe	Yes
Include	*<data_source_util.h >*
Parameters	`base`—Base address of the control and status slave.
Returns	`1`—Test pattern generator core is busy. `0`—Test pattern generator core is not busy.
Description	Checks if the test pattern generator is busy. The test pattern generator core is busy when it is sending data or has data in the command FIFO to be sent.

14.4.4.12 data_source_fill_level()

Table 218. data_source_fill_level()

Information Type	Description
Prototype	`int data_source_fill_level(alt_u32 base);`
Thread-safe	Yes
Include	*<data_source_util.h >*
Parameters	`base`—Base address of the control and status slave.
Returns	Number of commands in the command FIFO.
Description	Retrieves the number of commands currently in the command FIFO.

14.4.4.13 data_source_send_data()

Table 219. data_source_send_data()

Information Type	Description
Prototype	`int data_source_send_data(alt_u32 cmd_base, alt_u16 channel, alt_u16 size, alt_u32 flags, alt_u16 error, alt_u8 data_error_mask);`
Thread-safe	No
Include	*<data_source_util.h >*
Parameters	`cmd_base`—Base address of the command slave. `channel`—Channel to send the data. `size`—Data size. `flags` —Specifies whether to send or suppress SOP and EOP signals. Valid values are `DATA_SOURCE_SEND_SOP`, `DATA_SOURCE_SEND_EOP`, `DATA_SOURCE_SEND_SUPRESS_SOP` and `DATA_SOURCE_SEND_SUPRESS_EOP`. `error`—Value asserted on the `error` signal on the output interface. `data_error_mask`—Parameter and the data are XORed together to produce erroneous data.
Returns	Returns 1.
Description	Sends a data fragment to the specified channel. If data packets are supported, applications must ensure consistent usage of SOP and EOP in each channel. Except for the last segment in a packet, the length of each segment is a multiple of the data width. If data packets are not supported, applications must ensure that there are no SOP and EOP indicators in the data. The length of each segment in a packet is a multiple of the data width.

14.4.5 Test Pattern Checker API

The following subsections describe API for the test pattern checker core. The API functions are currently not available from the ISR.

data_sink_reset() on page 926

data_sink_init() on page 926

data_sink_get_id() on page 927

data_sink_get_supports_packets() on page 927

14.4.5.1 data_sink_reset()

Table 220. data_sink_reset()

Information Type	Description
Prototype	`void data_sink_reset(alt_u32 base);`
Thread-safe	No
Include	*<data_sink_util.h >*
Parameters	base—Base address of the control and status slave.
Returns	void
Description	Resets the test pattern checker core including all internal counters.

14.4.5.2 data_sink_init()

Table 221. data_sink_init()

Information Type	Description
Prototype	`int data_source_init(alt_u32 base);`
Thread-safe	No
Include	*<data_sink_util.h >*
Parameters	base—Base address of the control and status slave.
Returns	1—Initialization is successful.

continued...

Information Type	Description
	0—Initialization is unsuccessful.
Description	Performs the following operations to initialize the test pattern checker core: • Resets and disables the test pattern checker core. • Sets the throttle to the maximum value.

14.4.5.3 data_sink_get_id()

Table 222. data_sink_get_id()

Information Type	Description
Prototype	`int data_sink_get_id(alt_u32 base);`
Thread-safe	Yes
Include	*<data_sink_util.h >*
Parameters	base—Base address of the control and status slave.
Returns	Test pattern checker core identifier.
Description	Retrieves the test pattern checker core's identifier.

14.4.5.4 data_sink_get_supports_packets()

Table 223. data_sink_get_supports_packets()

Information Type	Description
Prototype	`int data_sink_init(alt_u32 base);`
Thread-safe	Yes
Include	*<data_sink_util.h >*
Parameters	base—Base address of the control and status slave.
Returns	1—Data packets are supported. 0—Data packets are not supported.
Description	Checks if the test pattern checker core supports data packets.

14.4.5.5 data_sink_get_num_channels()

Table 224. data_sink_get_num_channels()

Information Type	Description
Prototype	`int data_sink_get_num_channels(alt_u32 base);`
Thread-safe	Yes
Include	*<data_sink_util.h >*
Parameters	base—Base address of the control and status slave.
Returns	Number of channels supported.
Description	Retrieves the number of channels supported by the test pattern checker core.

14.4.5.6 data_sink_get_symbols_per_cycle()

Table 225. data_sink_get_symbols_per_cycle()

Information Type	Description
Prototype	`int data_sink_get_symbols(alt_u32 base);`
Thread-safe	Yes
Include	*<data_sink_util.h >*
Parameters	`base`—Base address of the control and status slave.
Returns	Number of symbols received in a beat.
Description	Retrieves the number of symbols received by the test pattern checker core in each beat.

14.4.5.7 data_sink_get_enable()

Table 226. data_sink_get_enable()

Information Type	Description
Prototype	`int data_sink_get_enable(alt_u32 base);`
Thread-safe	Yes
Include	*<data_sink_util.h >*
Parameters	`base`—Base address of the control and status slave.
Returns	Value of the ENABLE bit.
Description	Retrieves the value of the ENABLE bit.

14.4.5.8 data_sink_set enable()

Table 227. data_sink_set enable()

Information Type	Description
Prototype	`void data_sink_set_enable(alt_u32 base, alt_u32 value);`
Thread-safe	No
Include	*<data_sink_util.h >*
Parameters	`base`—Base address of the control and status slave. `value`—ENABLE bit is set to the value of the parameter.
Returns	`void`
Description	Enables the test pattern checker core.

14.4.5.9 data_sink_get_throttle()

Table 228. data_sink_get_throttle()

Information Type	Description
Prototype	`int data_sink_get_throttle(alt_u32 base);`
Thread-safe	Yes

continued...

Information Type	Description
Include	*<data_sink_util.h >*
Parameters	base—Base address of the control and status slave.
Returns	Throttle value.
Description	Retrieves the throttle value.

14.4.5.10 data_sink_set_throttle()

Table 229. data_sink_set_throttle()

Information Type	Description
Prototype	`void data_sink_set_throttle(alt_u32 base, alt_u32 value);`
Thread-safe	No
Include:	*<data_sink_util.h >*
Parameters	base—Base address of the control and status slave. value—Throttle value.
Returns	void
Description	Sets the throttle value, which can be between 0–256 inclusively. The throttle value, when divided by 256 yields the rate at which the test pattern checker receives data.

14.4.5.11 data_sink_get_packet_count()

Table 230. data_sink_get_packet_count()

Information Type	Description
Prototype	`int data_sink_get_packet_count(alt_u32 base, alt_u32 channel);`
Thread-safe	No
Include	*<data_sink_util.h >*
Parameters	base—Base address of the control and status slave. channel—Channel number.
Returns	Number of data packets received on the channel.
Description	Retrieves the number of data packets received on a channel.

14.4.5.12 data_sink_get_error_count()

Table 231. data_sink_get_error_count()

Information Type	Description
Prototype	`int data_sink_get_error_count(alt_u32 base, alt_u32 channel);`
Thread-safe	No
Include	*<data_sink_util.h >*
Parameters	base—Base address of the control and status slave.

continued...

Information Type	Description
	channel—Channel number.
Returns	Number of errors received on the channel.
Description	Retrieves the number of errors received on a channel.

14.4.5.13 data_sink_get_symbol_count()

Table 232. data_sink_get_symbol_count()

Information Type	Description
Prototype	`int data_sink_get_symbol_count(alt_u32 base, alt_u32 channel);`
Thread-safe	No
Include	*<data_sink_util.h >*
Parameters	base—Base address of the control and status slave. channel—Channel number.
Returns	Number of symbols received on the channel.
Description	Retrieves the number of symbols received on a channel.

14.4.5.14 data_sink_get_exception()

Table 233. data_sink_get_exception()

Information Type	Description
Prototype	`int data_sink_get_exception(alt_u32 base);`
Thread-safe	Yes
Include	*<data_sink_util.h >*
Parameters	base—Base address of the control and status slave.
Returns	First exception descriptor in the exception FIFO. 0—No exception descriptor found in the exception FIFO.
Description	Retrieves the first exception descriptor in the exception FIFO and pops it off the FIFO.

14.4.5.15 data_sink_exception_is_exception()

Table 234. data_sink_exception_is_exception()

Information Type	Description
Prototype	`int data_sink_exception_is_exception(int exception);`
Thread-safe	Yes
Include	*<data_sink_util.h >*
Parameters	exception—Exception descriptor
Returns	1—Indicates an exception. 0—No exception.
Description	Checks if an exception descriptor describes a valid exception.

14.4.5.16 data_sink_exception_has_data_error()

Table 235. data_sink_exception_has_data_error()

Information Type	Description
Prototype	`int data_sink_exception_has_data_error(int exception);`
Thread-safe	Yes
Include	*<data_sink_util.h >*
Parameters	`exception`—Exception descriptor.
Returns	`1`—Data has errors. `0`—No errors.
Description	Checks if an exception indicates erroneous data.

14.4.5.17 data_sink_exception_has_missing_sop()

Table 236. data_sink_exception_has_missing_sop()

Information Type	Description
Prototype	`int data_sink_exception_has_missing_sop(int exception);`
Thread-safe	Yes
Include	*<data_sink_util.h >*
Parameters	`exception`—Exception descriptor.
Returns	`1`—Missing SOP. `0`—Other exception types.
Description	Checks if an exception descriptor indicates missing SOP.

14.4.5.18 data_sink_exception_has_missing_eop()

Table 237. data_sink_exception_has_missing_eop()

Information Type	Description
Prototype	`int data_sink_exception_has_missing_eop(int exception);`
Thread-safe	Yes
Include	*<data_sink_util.h >*
Parameters	`exception`—Exception descriptor.
Returns	`1`—Missing EOP. `0`—Other exception types.
Description	Checks if an exception descriptor indicates missing EOP.

14.4.5.19 data_sink_exception_signalled_error()

Table 238. data_sink_exception_signalled_error()

Information Type	Description
Prototype	`int data_sink_exception_signalled_error(int exception);`
Thread-safe	Yes
Include	*<data_sink_util.h >*
Parameters	`exception`—Exception descriptor.
Returns	Signal error value.
Description	Retrieves the value of the signaled error from the exception.

14.4.5.20 data_sink_exception_channel()

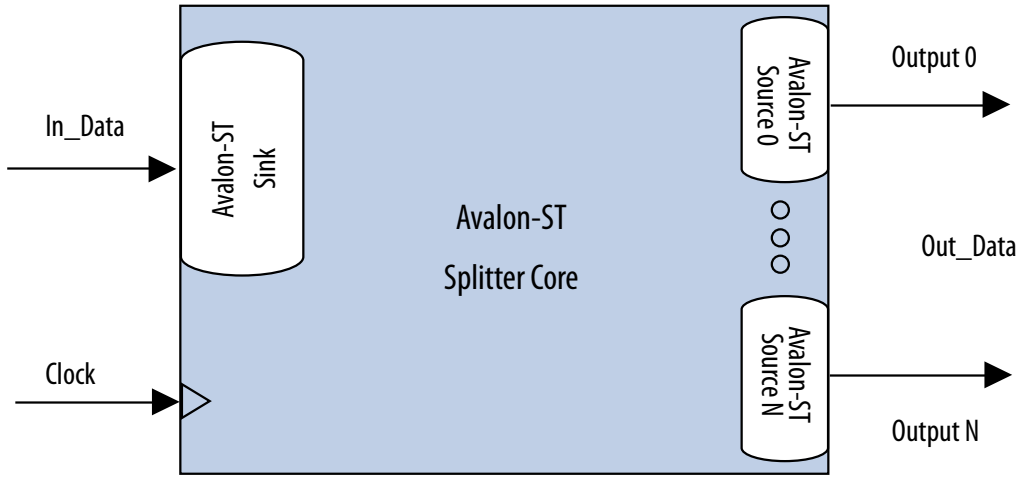
Table 239. data_sink_exception_channel()

Information Type	Description
Prototype	`int data_sink_exception_channel(int exception);`
Thread-safe	Yes
Include	*<data_sink_util.h >*
Parameters	`exception`—Exception descriptor.
Returns	Channel number on which an exception occurred.
Description	Retrieves the channel number on which an exception occurred.

14.5 Avalon-ST Splitter Core

Figure 272. Avalon-ST Splitter Core

The Avalon-ST Splitter Core allows you to replicate transactions from an Avalon-ST sink interface to multiple Avalon-ST source interfaces. This core supports from 1 to 16 outputs.



The Avalon-ST Splitter core copies input signals from the input interface to the corresponding output signals of each output interface without altering the size or functionality. This includes all signals except for the `ready` signal. The core includes a clock signal to determine the Avalon-ST interface and clock domain where the core resides. Because the splitter core does not use the `clock` signal internally, latency is not introduced when using this core.

14.5.1 Splitter Core Backpressure

The Avalon-ST Splitter core integrates with backpressure by AND-ing the `ready` signals from the output interfaces and sending the result to the input interface. As a result, if an output interface deasserts the `ready` signal, the input interface receives the deasserted `ready` signal, as well. This functionality ensures that backpressure on the output interfaces is propagated to the input interface.

When the **Qualify Valid Out** option is enabled, the `out_valid` signals on all other output interfaces are gated when backpressure is applied from one output interface. In this case, when any output interface deasserts its `ready` signal, the `out_valid` signals on the other output interfaces are also deasserted.

When the **Qualify Valid Out** option is disabled, the output interfaces have a non-gated `out_valid` signal when backpressure is applied. In this case, when an output interface deasserts its ready signal, the `out_valid` signals on the other output interfaces are not affected.

Because the logic is combinational, the core introduces no latency.

14.5.2 Splitter Core Interfaces

The Avalon-ST Splitter core supports streaming data, with optional packet, channel, and error signals. The core propagates backpressure from any output interface to the input interface.

Table 240. Avalon-ST Splitter Core Support

Feature	Support
Backpressure	Ready latency = 0.
Data Width	Configurable.
Channel	Supported (optional).
Error	Supported (optional).
Packet	Supported (optional).

14.5.3 Splitter Core Parameters

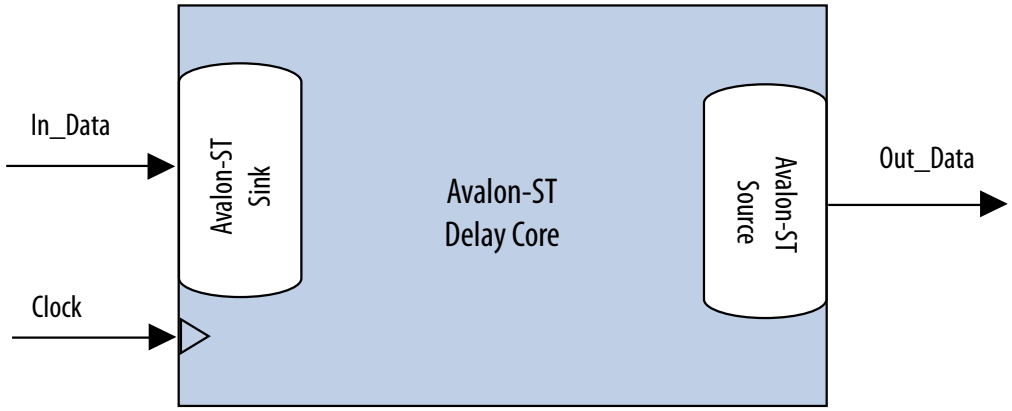
Table 241. Avalon-ST Splitter Core Parameters

Parameter	Legal Values	Default Value	Description
Number Of Outputs	1 to 16	2	The number of output interfaces. Qsys Pro supports 1 for some systems where no duplicated output is required.
Qualify Valid Out	Enabled, Disabled	Enabled	If enabled, the `out_valid` signal of all output interfaces is gated when back pressure is applied.
Data Width	1–512	8	The width of the data on the Avalon-ST data interfaces.
Bits Per Symbol	1–512	8	The number of bits per symbol for the input and output interfaces. For example, byte-oriented interfaces have 8-bit symbols.
Use Packets	Enabled, Disabled	Disabled	Enable support of data packet transfers. Packet support includes the `startofpacket`, `endofpacket`, and `empty` signals.
Use Channel	Enabled, Disabled	Disabled	Enable the channel signal.
Channel Width	0-8	1	The width of the `channel` signal on the data interfaces. This parameter is disabled when **Use Channel** is set to 0.
Max Channels	0-255	1	The maximum number of channels that a data interface can support. This parameter is disabled when **Use Channel** is set to 0.
Use Error	Enabled, Disabled	Disabled	Enable the error signal.
Error Width	0–31	1	The width of the `error` signal on the output interfaces. A value of `0` indicates that the splitter core is not using the `error` signal. This parameter is disabled when **Use Error** is set to `0`.

14.6 Avalon-ST Delay Core

Figure 273. Avalon-ST Delay Core

The Avalon-ST Delay Core provides a solution to delay Avalon-ST transactions by a constant number of clock cycles. This core supports up to 16 clock cycle delays.



The Avalon-ST Delay core adds a delay between the input and output interfaces. The core accepts transactions presented on the input interface and reproduces them on the output interface N cycles later without changing the transaction.

The input interface delays the input signals by a constant N number of clock cycles to the corresponding output signals of the output interface. The **Number Of Delay Clocks** parameter defines the constant N, which must be from 0 to 16. The change of the `in_valid` signal is reflected on the `out_valid` signal exactly N cycles later.

14.6.1 Delay Core Reset Signal

The Avalon-ST Delay core has a `reset` signal that is synchronous to the `clk` signal. When the core asserts the `reset` signal, the output signals are held at 0. After the `reset` signal is deasserted, the output signals are held at 0 for N clock cycles. The delayed values of the input signals are then reflected at the output signals after N clock cycles.

14.6.2 Delay Core Interfaces

The Delay core supports streaming data, with optional packet, channel, and error signals. The delay core does not support backpressure.

Table 242. Avalon-ST Delay Core Support

Feature	Support
Backpressure	Not supported.
Data Width	Configurable.
Channel	Supported (optional).
Error	Supported (optional).
Packet	Supported (optional).

14.6.3 Delay Core Parameters

Table 243. Avalon-ST Delay Core Parameters

Parameter	Legal Values	Default Value	Description
Number Of Delay Clocks	0 to 16	1	Specifies the delay the core introduces, in clock cycles. Qsys Pro supports 0 for some systems where no delay is required.
Data Width	1–512	8	The width of the data on the Avalon-ST data interfaces.
Bits Per Symbol	1–512	8	The number of bits per symbol for the input and output interfaces. For example, byte-oriented interfaces have 8-bit symbols.
Use Packets	0 or 1	0	Indicates whether or not data packet transfers are supported. Packet support includes the `startofpacket`, `endofpacket`, and `empty` signals.
Use Channel	0 or 1	0	The option to enable or disable the channel signal.

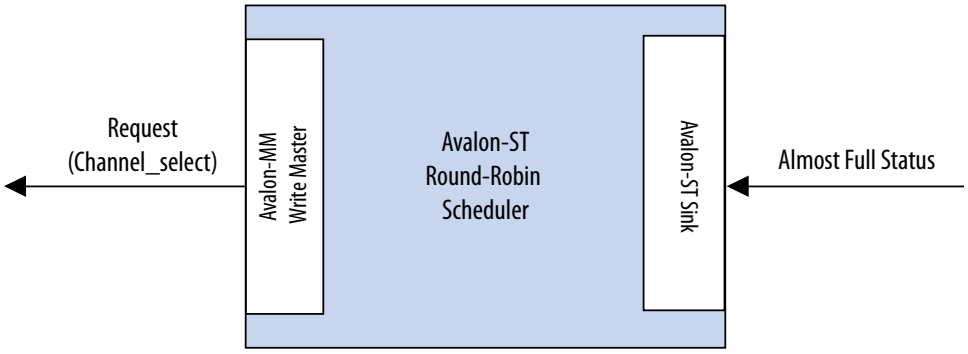
continued...

Parameter	Legal Values	Default Value	Description
Channel Width	0-8	1	The width of the `channel` signal on the data interfaces. This parameter is disabled when **Use Channel** is set to 0.
Max Channels	0-255	1	The maximum number of channels that a data interface can support. This parameter is disabled when **Use Channel** is set to 0.
Use Error	0 or 1	0	The option to enable or disable the error signal.
Error Width	0–31	1	The width of the `error` signal on the output interfaces. A value of 0 indicates that the error signal is not in use. This parameter is disabled when **Use Error** is set to 0.

14.7 Avalon-ST Round Robin Scheduler

Figure 274. Avalon-ST Round Robin Scheduler

The Avalon-ST Round Robin Scheduler core controls the read operations from a multi-channel Avalon-ST component that buffers data by channels. It reads the almost-full threshold values from the multiple channels in the multi-channel component and issues the read request to the Avalon-ST source according to a round-robin scheduling algorithm.



In a multi-channel component, the component can store data either in the sequence that it comes in (FIFO), or in segments according to the channel. When data is stored in segments according to channels, a scheduler is needed to schedule the read operations.

14.7.1 Almost-Full Status Interface (Round Robin Scheduler)

The Almost-Full Status interface is an Avalon-ST sink interface that collects the almost-full status from the sink components for the channels in the sequence provided.

Table 244. Avalon-ST Interface Feature Support

Feature	Property
Backpressure	Not supported
Data Width	Data width = 1; Bits per symbol = 1
	continued...

Feature	Property
Channel	Maximum channel = 32; Channel width = 5
Error	Not supported
Packet	Not supported

14.7.2 Request Interface (Round Robin Scheduler)

The Request Interface is an Avalon-MM write master interface that requests data from a specific channel. The Avalon-ST Round Robin Scheduler cycles through the channels it supports and schedules data to be read.

14.7.3 Round Robin Scheduler Operation

If a particular channel is almost full, the Avalon-ST Round Robin Scheduler does not schedule data to be read from that channel in the source component.

The scheduler only requests 1 bit of data from a channel at each transaction. To request 1 bit of data from channel *n*, the scheduler writes the value 1 to address (4 ×*n*). For example, if the scheduler is requesting data from channel 3, the scheduler writes 1 to address `0xC`. At every clock cycle, the scheduler requests data from the next channel. Therefore, if the scheduler starts requesting from channel 1, at the next clock cycle, it requests from channel 2. The scheduler does not request data from a particular channel if the almost-full status for the channel is asserted. In this case, the scheduler uses one clock cycle without a request transaction.

The Avalon-ST Round Robin Scheduler cannot determine if the requested component is able to service the request transaction. The component asserts `waitrequest` when it cannot accept new requests.

Table 245. Avalon-ST Round Robin Scheduler Ports

Signal	Direction	Description
Clock and Reset		
`clk`	In	Clock reference.
`reset_n`	In	Asynchronous active low reset.
Avalon-MM Request Interface		
`request_address (log2 Max_Channels-1:0)`	Out	The write address that indicates which channel has the request.
`request_write`	Out	Write enable signal.
`request_writedata`	Out	The amount of data requested from the particular channel. This value is always fixed at 1.
`request_waitrequest`	In	Wait request signal that pauses the scheduler when the slave cannot accept a new request.
Avalon-ST Almost-Full Status Interface		

continued...

Signal	Direction	Description
almost_full_valid	In	Indicates that almost_full_channel and almost_full_data are valid.
almost_full_channel (Channel_Width-1:0)	In	Indicates the channel for the current status indication.
almost_full_data (log_2 Max_Channels-1:0)	In	A 1-bit signal that is asserted high to indicate that the channel indicated by almost_full_channel is almost full.

14.7.4 Round Robin Scheduler Parameters

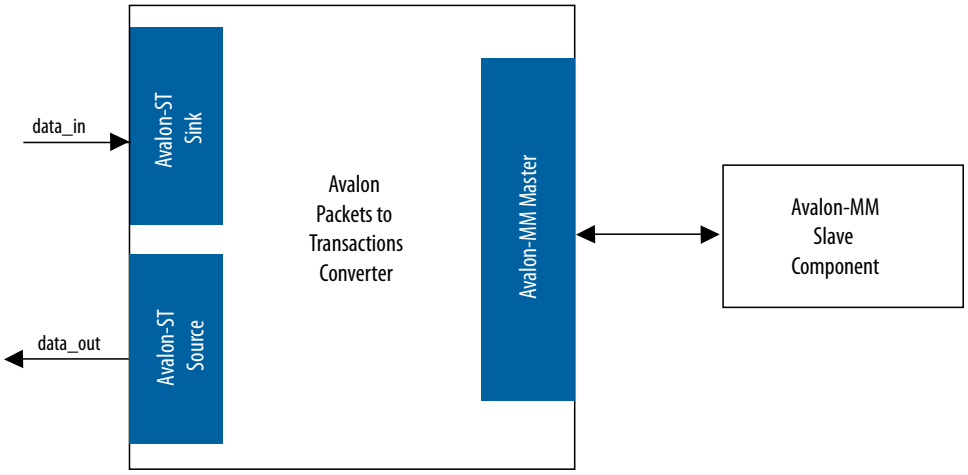
Table 246. Avalon-ST Round Robin Scheduler Parameters

Parameters	Legal Values	Default Value	Description
Number of channels	2–32	2	Specifies the number of channels the Avalon-ST Round Robin Scheduler supports.
Use almost-full status	Enabled, Disabled	Disabled	If enabled, the scheduler uses the almost-full interface. If not, the core requests data from the next channel at the next clock cycle.

14.8 Avalon Packets to Transactions Converter

Figure 275. Avalon Packets to Transactions Converter Core

The Avalon Packets to Transactions Converter core receives streaming data from upstream components and initiates Avalon-MM transactions. The core then returns Avalon-MM transaction responses to the requesting components.



Note: The SPI Slave to Avalon Master Bridge and JTAG to Avalon Master Bridge are examples of the Packets to Transactions Converter core. For more information, refer to the *Avalon Interface Specifications*.

Related Links

Avalon Interface Specifications

14.8.1 Packets to Transactions Converter Interfaces

Table 247. Properties of Avalon-ST Interfaces

Feature	Property
Backpressure	Ready latency = 0.
Data Width	Data width = 8 bits; Bits per symbol = 8.
Channel	Not supported.
Error	Not used.
Packet	Supported.

The Avalon-MM master interface supports read and write transactions. The data width is set to 32 bits, and burst transactions are not supported.

14.8.2 Packets to Transactions Converter Operation

The Packets to Transactions Converter core receives streams of packets on its Avalon-ST sink interface and initiates Avalon-MM transactions. Upon receiving transaction responses from Avalon-MM slaves, the core transforms the responses to packets and returns them to the requesting components via its Avalon-ST source interface. The core does not report Avalon-ST errors.

14.8.2.1 Packets to Transactions Converter Data Packet Formats

A response packet is returned for every write transaction. The core also returns a response packet if a no transaction (`0x7f`) is received. An invalid transaction code is regarded as a no transaction. For read transactions, the core returns the data read.

The Packets to Transactions Converter core expects incoming data streams to be in the formats shown in the table below.

Table 248. Data Packet Formats

Byte	Field	Description
Transaction Packet Format		
0	`Transaction code`	Type of transaction.
1	Reserved	Reserved for future use.
[3:2]	`Size`	Transaction size in bytes. For write transactions, the size indicates the size of the `data` field. For read transactions, the size indicates the total number of bytes to read.
[7:4]	`Address`	32-bit address for the transaction.
[n:8]	`Data`	Transaction data; data to be written for write transactions.
Response Packet Format		
0	`Transaction code`	The transaction code with the most significant bit inversed.
1	Reserved	Reserved for future use.
[4:2]	`Size`	Total number of bytes read/written successfully.

Related Links

Packets to Transactions Converter Interfaces on page 939

14.8.2.2 Packets to Transactions Converter Supported Transactions

Table 249. Packets to Transactions Converter Supported Transactions

Avalon-MM transactions supported by the Packets to Transactions Converter core.

Transaction Code	Avalon-MM Transaction	Description
0x00	Write, non-incrementing address.	Writes data to the address until the total number of bytes written to the same word address equals to the value specified in the size field.
0x04	Write, incrementing address.	Writes transaction data starting at the current address.
0x10	Read, non-incrementing address.	Reads 32 bits of data from the address until the total number of bytes read from the same address equals to the value specified in the size field.
0x14	Read, incrementing address.	Reads the number of bytes specified in the size parameter starting from the current address.
0x7f	No transaction.	No transaction is initiated. You can use this transaction type for testing purposes. Although no transaction is initiated on the Avalon-MM interface, the core still returns a response packet for this transaction code.

The Packets to Transactions Converter core can process only a single transaction at a time. The ready signal on the core's Avalon-ST sink interface is asserted only when the current transaction is completely processed.

No internal buffer is implemented on the datapaths. Data received on the Avalon-ST interface is forwarded directly to the Avalon-MM interface and vice-versa. Asserting the waitrequest signal on the Avalon-MM interface backpressures the Avalon-ST sink interface. In the opposite direction, if the Avalon-ST source interface is backpressured, the read signal on the Avalon-MM interface is not asserted until the backpressure is alleviated. Backpressuring the Avalon-ST source in the middle of a read could result in data loss. In this cases, the core returns the data that is successfully received.

A transaction is considered complete when the core receives an EOP. For write transactions, the actual data size is expected to be the same as the value of the size property. Whether or not both values agree, the core always uses the end of packet (EOP) to determine the end of data.

14.8.2.3 Packets to Transactions Converter Malformed Packets

The following are examples of malformed packets:

- **Consecutive start of packet (SOP)**—An SOP marks the beginning of a transaction. If an SOP is received in the middle of a transaction, the core drops the current transaction without returning a response packet for the transaction, and initiates a new transaction. This effectively precesses packets without an end of packet (EOP).
- **Unsupported transaction codes**—The core processes unsupported transactions as a no transaction.

14.9 Avalon-ST Streaming Pipeline Stage

The Avalon-ST pipeline stage receives data from an Avalon-ST source interface, and outputs the data to an Avalon-ST sink interface. In the absence of back pressure, the Avalon-ST pipeline stage source interface outputs data one cycle after receiving the data on its sink interface.

If the pipeline stage receives back pressure on its source interface, it continues to assert its source interface's current data output. While the pipeline stage is receiving back pressure on its source interface and it receives new data on its sink interface, the pipeline stage internally buffers the new data. It then asserts back pressure on its sink interface.

After the backpressure is deasserted, the pipeline stage's source interface is deasserted and the pipeline stage asserts internally buffered data (if present). Additionally, the pipeline stage deasserts back pressure on its sink interface.

Figure 276. Pipeline Stage Simple Register

If the ready signal is not pipelined, the pipeline stage becomes a simple register.

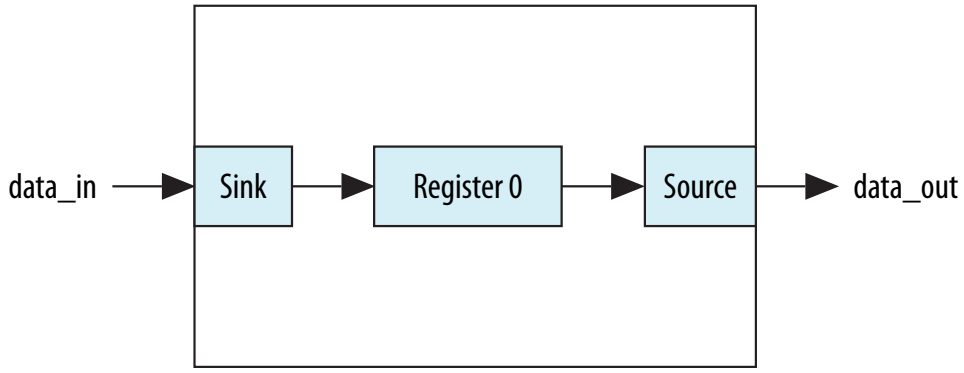
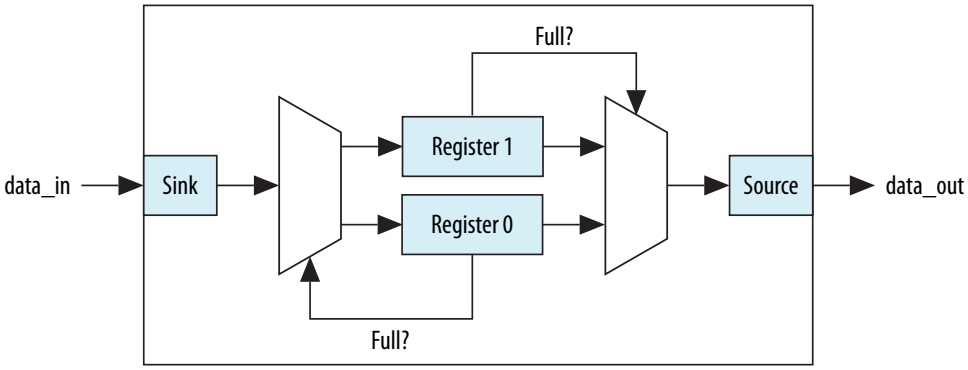


Figure 277. Pipeline Stage Holding Register

If the ready signal is pipelined, the pipeline stage must also include a second "holding" register.



14.10 Streaming Channel Multiplexer and Demultiplexer Cores

The Avalon-ST channel multiplexer core receives data from various input interfaces and multiplexes the data into a single output interface, using the optional `channel` signal to indicate the origin of the data. The Avalon-ST channel demultiplexer core receives data from a channelized input interface and drives that data to multiple output interfaces, where the output interface is selected by the input `channel` signal.

The multiplexer and demultiplexer cores can transfer data between interfaces on cores that support unidirectional flow of data. The multiplexer and demultiplexer allow you to create multiplexed or demultiplexed datapaths without having to write custom HDL code. The multiplexer includes an Avalon-ST Round Robin Scheduler.

Related Links

Avalon-ST Round Robin Scheduler on page 936

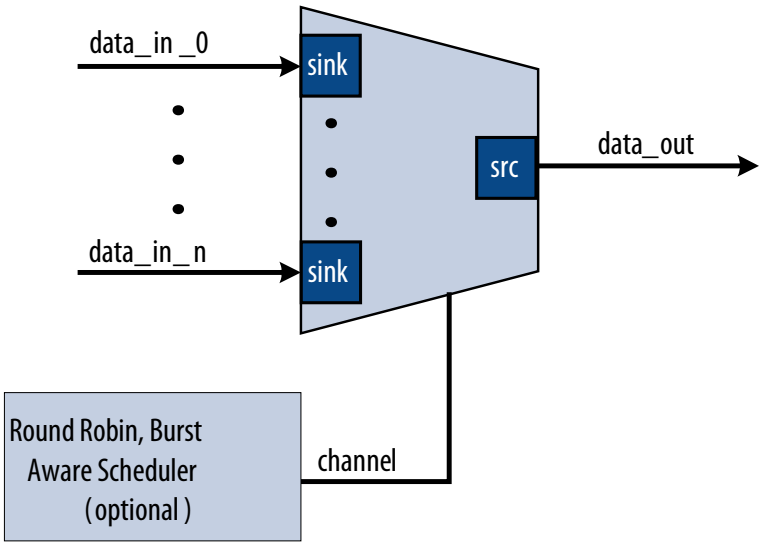
14.10.1 Software Programming Model For the Multiplexer and Demultiplexer Components

The multiplexer and demultiplexer components do not have any user-visible control or status registers. Therefore, Qsys Pro cannot control or configure any aspect of the multiplexer or demultiplexer at run-time. The components cannot generate interrupts.

14.10.2 Avalon-ST Multiplexer

Figure 278. Avalon-ST Multiplexer

The Avalon-ST multiplexer takes data from a variety of input data interfaces, and multiplexes the data onto a single output interface. The multiplexer includes a round-robin scheduler that selects from the next input interface that has data. Each input interface has the same width as the output interface, so that the other input interfaces are backpressured when the multiplexer is carrying data from a different input interface.



The multiplexer includes an optional channel signal that enables each input interface to carry channelized data. The output interface channel width is equal to:

(log2 (n-1)) + 1 + w

where n is the number of input interfaces, and w is the channel width of each input interface. All input interfaces must have the same channel width. These bits are appended to either the most or least significant bits of the output channel signal.

The scheduler processes one input interface at a time, selecting it for transfer. Once an input interface has been selected, data from that input interface is sent until one of the following scenarios occurs:

- The specified number of cycles have elapsed.
- The input interface has no more data to send and the `valid` signal is deasserted on a ready cycle.
- When packets are supported, `endofpacket` is asserted.

14.10.2.1 Multiplexer Input Interfaces

Each input interface is an Avalon-ST data interface that optionally supports packets. The input interfaces are identical; they have the same symbol and data widths, error widths, and channel widths.

14.10.2.2 Multiplexer Output Interface

The output interface carries the multiplexed data stream with data from the inputs. The symbol, data, and error widths are the same as the input interfaces.

The width of the `channel` signal is the same as the input interfaces, with the addition of the bits needed to indicate the origin of the data.

You can configure the following parameters for the output interface:

- **Data Bits Per Symbol**—The bits per symbol is related to the width of `readdata` and `writedata` signals, which must be a multiple of the bits per symbol.
- **Data Symbols Per Beat**—The number of symbols (words) that are transferred per beat (transfer). Valid values are 1 to 32.
- **Include Packet Support**—Indicates whether or not packet transfers are supported. Packet support includes the `startofpacket`, `endofpacket`, and `empty` signals.
- **Channel Signal Width (bits)**— The number of bits Qsys Pro uses for the channel signal for output interfaces. For example, set this parameter to 1 if you have two input interfaces with no channel, or set this parameter to 2 if you have two input interfaces with a channel width of 1 bit. The input channel can have a width between 0-31 bits.
- **Error Signal Width (bits)**—The width of the `error` signal for input and output interfaces. A value of `0` means the `error` signal is not in use.

Note: If you change only bits per symbol, and do not change the data width, errors are generated.

14.10.2.3 Multiplexer Parameters

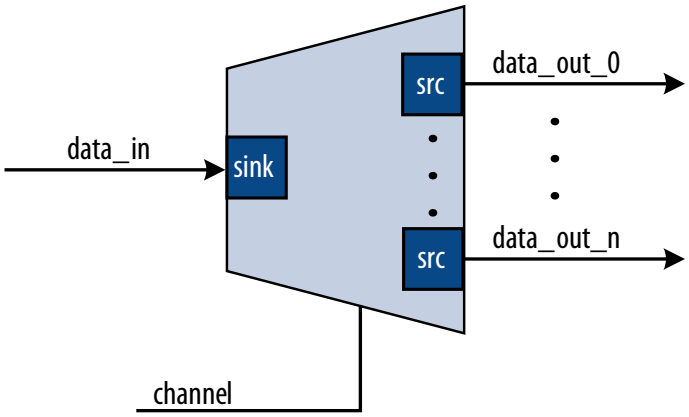
You can configure the following parameters for the multiplexer:

- **Number of Input Ports**—The number of input interfaces that the multiplexer supports. Valid values are 2 to 16.
- **Scheduling Size (Cycles)**—The number of cycles that are sent from a single channel before changing to the next channel.
- **Use Packet Scheduling**—When this parameter is turned on, the multiplexer only switches the selected input interface on packet boundaries. Therefore, packets on the output interface are not interleaved.
- **Use high bits to indicate source port**—When this parameter is turned on, the multiplexer uses the high bits of the output `channel` signal to indicate the origin of the input interface of the data. For example, if the input interfaces have 4-bit channel signals, and the multiplexer has 4 input interfaces, the output interface has a 6-bit channel signal. If this parameter is turned on, bits [5:4] of the output channel signal indicate origin of the input interface of the data, and bits [3:0] are the channel bits that were presented at the input interface.

14.10.3 Avalon-ST Demultiplexer

Figure 279. Avalon-ST Demultiplexer

That Avalon-ST demultiplexer takes data from a channelized input data interface and provides that data to multiple output interfaces, where the output interface selected for a particular transfer is specified by the input `channel` signal.



The data is delivered to the output interfaces in the same order it is received at the input interface, regardless of the value of `channel`, `packet`, `frame`, or any other signal. Each of the output interfaces has the same width as the input interface; each output interface is idle when the demultiplexer is driving data to a different output interface. The demultiplexer uses $\log_2$ (`num_output_interfaces`) bits of the `channel` signal to select the output for the data; the remainder of the channel bits are forwarded to the appropriate output interface unchanged.

14.10.3.1 Demultiplexer Input Interface

Each input interface is an Avalon-ST data interface that optionally supports packets. You can configure the following parameters for the input interface:

- **Data Bits Per Symbol**—The bits per symbol is related to the width of `readdata` and `writedata` signals, which must be a multiple of the bits per symbol.
- **Data Symbols Per Beat**—The number of symbols (words) that are transferred per beat (transfer). Valid values are 1 to 32.
- **Include Packet Support**—Indicates whether or not data packet transfers are supported. Packet support includes the `startofpacket`, `endofpacket`, and `empty` signals.
- **Channel Signal Width (bits)**—The number of bits for the `channel` signal for output interfaces. A value of `0` means that output interfaces do not use the optional `channel` signal.
- **Error Signal Width (bits)**—The width of the `error` signal for input and output interfaces. A value of `0` means the `error` signal is in use.

Note: If you change only bits per symbol, and do not change the data width, errors are generated.

14.10.3.2 Demultiplexer Output Interface

Each output interface carries data from a subset of channels from the input interface. Each output interface is identical; all have the same symbol and data widths, error widths, and channel widths. The symbol, data, and error widths are the same as the input interface. The width of the `channel` signal is the same as the input interface, without the bits that the demultiplexer uses to select the output interface.

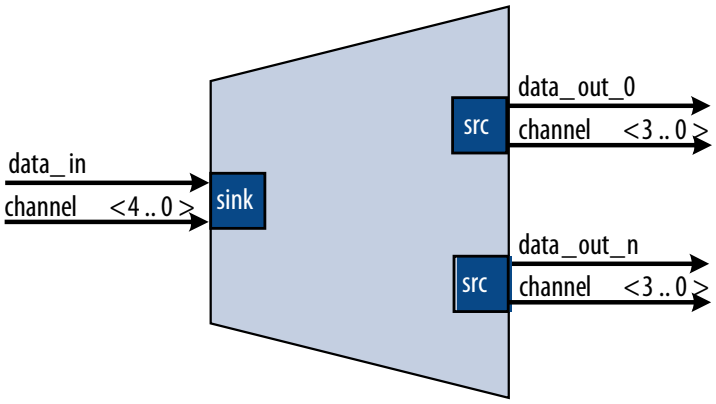
14.10.3.3 Demultiplexer Parameters

You can configure the following parameters for the demultiplexer:

- **Number of Output Ports**—The number of output interfaces that the multiplexer supports Valid values are 2 to 16.
- **High channel bits select output**—When this option is turned on, the demultiplexing function uses the high bits of the input `channel` signal, and the low order bits are passed to the output. When this option is turned off, the demultiplexing function uses the low order bits, and the high order bits are passed to the output.

Where you place the signals in your design affects the functionality; for example, there is one input interface and two output interfaces. If the low-order bits of the channel signal select the output interfaces, the even channels go to channel 0, and the odd channels go to channel 1. If the high-order bits of the channel signal select the output interface, channels 0 to 7 go to channel 0 and channels 8 to 15 go to channel 1.

Figure 280. Select Bits for the Demultiplexer



14.11 Single-Clock and Dual-Clock FIFO Cores

The Avalon-ST Single-Clock and Avalon-ST Dual-Clock FIFO cores are FIFO buffers which operate with a common clock and independent clocks for input and output ports respectively.

Figure 281. Avalon-ST Single Clock FIFO Core

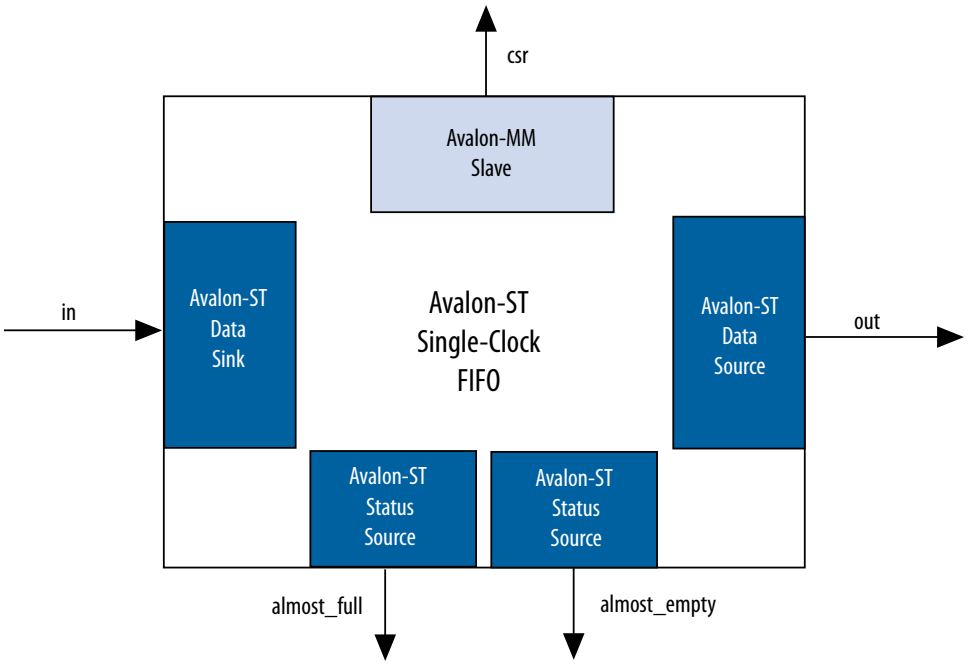
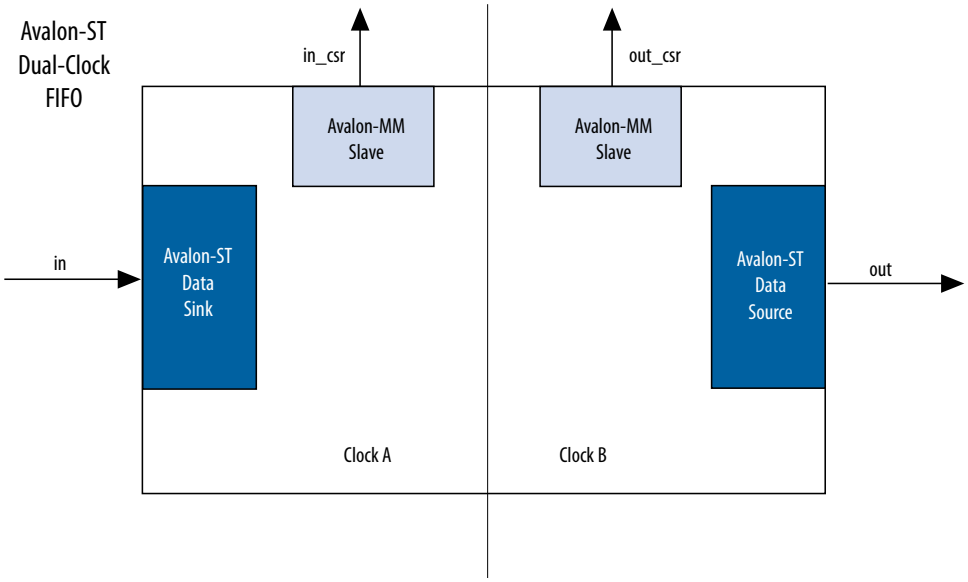


Figure 282. Avalon-ST Dual Clock FIFO Core



14.11.1 Interfaces Implemented in FIFO Cores

The following interfaces are implemented in FIFO cores:

Avalon-ST Data Interface on page 947

Avalon-MM Control and Status Register Interface on page 947

Avalon-ST Status Interface on page 948

14.11.1.1 Avalon-ST Data Interface

Each FIFO core has an Avalon-ST data sink and source interfaces. The data sink and source interfaces in the dual-clock FIFO core are driven by different clocks.

Table 250. Avalon-ST Interfaces Properties

Feature	Property
Backpressure	Ready latency = 0.
Data Width	Configurable.
Channel	Supported, up to 255 channels.
Error	Configurable.
Packet	Configurable.

14.11.1.2 Avalon-MM Control and Status Register Interface

You can configure the single-clock FIFO core to include an optional Avalon-MM interface, and the dual-clock FIFO core to include an Avalon-MM interface in each clock domain. The Avalon-MM interface provides access to 32-bit registers, which allows you

to retrieve the FIFO buffer fill level and configure the almost-empty and almost-full thresholds. In the single-clock FIFO core, you can also configure the packet and error handling modes.

14.11.1.3 Avalon-ST Status Interface

The single-clock FIFO core has two optional Avalon-ST status source interfaces from which you can obtain the FIFO buffer almost-full and almost empty statuses.

14.11.2 FIFO Operating Modes

- **Default mode**—The core accepts incoming data on the `in` interface (Avalon-ST data sink) and forwards it to the `out` interface (Avalon-ST data source). The core asserts the `valid` signal on the Avalon-ST source interface to indicate that data is available at the interface.
- **Store and forward mode**—This mode applies only to the single-clock FIFO core. The core asserts the `valid` signal on the `out` interface only when a full packet of data is available at the interface. In this mode, you can also enable the drop-on-error feature by setting the `drop_on_error` register to 1. When this feature is enabled, the core drops all packets received with the `in_error` signal asserted.
- **Cut-through mode**—This mode applies only to the single-clock FIFO core. The core asserts the `valid` signal on the `out` interface to indicate that data is available for consumption when the number of entries specified in the `cut_through_threshold` register are available in the FIFO buffer.

Note: To turn on **Cut-through mode**, the **Use store and forward** parameter must be set to 0. Turning on **Use store and forward mode** prompts the user to turn on **Use fill level**, and then the CSR appears.

14.11.3 Fill Level of the FIFO Buffer

You can obtain the fill level of the FIFO buffer via the optional Avalon-MM control and status interface. Turn on the **Use fill level** parameter (**Use sink fill level** and **Use source fill level** in the dual-clock FIFO core) and read the `fill_level` register.

The dual-clock FIFO core has two fill levels, one in each clock domain. Due to the latency of the clock crossing logic, the fill levels reported in the input and output clock domains may be different for any instance. In both cases, the fill level may report badly for the clock domain; that is, the fill level is reported high in the input clock domain, and low in the output clock domain.

The dual-clock FIFO has an output pipeline stage to improve f_{MAX}. This output stage is accounted for when calculating the output fill level, but not when calculating the input fill level. Therefore, the best measure of the amount of data in the FIFO is by the fill level in the output clock domain. The fill level in the input clock domain represents the amount of space available in the FIFO (available space = FIFO depth – input fill level).

14.11.4 Almost-Full and Almost-Empty Thresholds to Prevent Overflow and Underflow

You can use almost-full and almost-empty thresholds as a mechanism to prevent FIFO overflow and underflow. This feature is available only in the single-clock FIFO core. To use the thresholds, turn on the **Use fill level**, **Use almost-full status**, and **Use**

almost-empty status parameters. You can access the `almost_full_threshold` and `almost_empty_threshold` registers via the csr interface and set the registers to an optimal value for your application.

You can obtain the almost-full and almost-empty statuses from `almost_full` and `almost_empty` interfaces (Avalon-ST status source). The core asserts the `almost_full` signal when the fill level is equal to or higher than the almost-full threshold. Likewise, the core asserts the `almost_empty` signal when the fill level is equal to or lower than the almost-empty threshold.

14.11.5 Single-Clock and Dual-Clock FIFO Core Parameters

Table 251. Single-Clock and Dual-Clock FIFO Core Parameters

Parameter	Legal Values	Description
Bits per symbol	1–32	These parameters determine the width of the FIFO. FIFO width = **Bits per symbol** * **Symbols per beat**, where: **Bits per symbol** is the number of bits in a symbol, and **Symbols per beat** is the number of symbols transferred in a beat.
Symbols per beat	1–32	
Error width	0–32	The width of the `error` signal.
FIFO depth	2^n	The FIFO depth. An output pipeline stage is added to the FIFO to increase performance, which increases the FIFO depth by one. <n> = n=1,2,3,4...
Use packets	—	Turn on this parameter to enable data packet support on the Avalon-ST data interfaces.
Channel width	1–32	The width of the `channel` signal.
Avalon-ST Single Clock FIFO Only		
Use fill level	—	Turn on this parameter to include the Avalon-MM control and status register interface (CSR). The CSR is enabled when **Use fill level** is set to 1.
Use Store and Forward		To turn on **Cut-through mode**, **Use store and forward** must be set to 0. Turning on **Use store and forward** prompts the user to turn on **Use fill level**, and then the CSR appears.
Avalon-ST Dual Clock FIFO Only		
Use sink fill level	—	Turn on this parameter to include the Avalon-MM control and status register interface in the input clock domain.
Use source fill level	—	Turn on this parameter to include the Avalon-MM control and status register interface in the output clock domain.
Write pointer synchronizer length	2–8	The length of the write pointer synchronizer chain. Setting this parameter to a higher value leads to better metastability while increasing the latency of the core.
Read pointer synchronizer length	2–8	The length of the read pointer synchronizer chain. Setting this parameter to a higher value leads to better metastability.
Use Max Channel	—	Turn on this parameter to specify the maximum channel number.
Max Channel	1–255	Maximum channel number.

Note: For more information on metastability in Intel devices, refer to *Understanding Metastability in FPGAs*. For more information on metastability analysis and synchronization register chains, refer to the *Managing Metastability*.

Related Links

- Managing Metastability with the Quartus Prime Software on page 952
  You can use the Quartus Prime software to analyze the average mean time between failures (MTBF) due to metastability caused by synchronization of asynchronous signals, and optimize the design to improve the metastability MTBF.
- Understanding Metastability in FPGAs

14.11.6 Avalon-ST Single-Clock FIFO Registers

Table 252. Avalon-ST Single-Clock FIFO Registers

The CSR interface in the Avalon-ST Single Clock FIFO core provides access to registers.

32-Bit Word Offset	Name	Access	Reset	Description
0	`fill_level`	R	0	24-bit FIFO fill level. Bits 24 to 31 are not used.
1	Reserved	—	—	Reserved for future use.
2	`almost_full_threshold`	RW	**FIFO depth**–1	Set this register to a value that indicates the FIFO buffer is getting full.
3	`almost_empty_threshold`	RW	0	Set this register to a value that indicates the FIFO buffer is getting empty.
4	`cut_through_threshold`	RW	0	**0**—Enables store and forward mode. **Greater than 0**—Enables cut-through mode and specifies the minimum of entries in the FIFO buffer before the `valid` signal on the Avalon-ST source interface is asserted. Once the FIFO core starts sending the data to the downstream component, it continues to do so until the end of the packet. *Note:* To turn on **Cut-through mode**, **Use store and forward** must be set to 0. Turning on **Use store and forward mode** prompts the user to turn on **Use fill level**, and then the CSR appears.
5	`drop_on_error`	RW	0	**0**—Disables drop-on error. **1**—Enables drop-on error. This register applies only when the **Use packet** and **Use store and forward** parameters are turned on.

Table 253. Register Description for Avalon-ST Dual-Clock FIFO

The `in_csr` and `out_csr` interfaces in the Avalon-ST Dual Clock FIFO core reports the FIFO fill level.

32-Bit Word Offset	Name	Access	Reset Value	Description
0	`fill_level`	R	0	24-bit FIFO fill level. Bits 24 to 31 are not used.

Related Links

- Avalon Interface Specifications
- Avalon Memory-Mapped Design Optimizations

14.12 Document Revision History

Table 254. Document Revision History

The table below indicates edits made to the *Qsys Pro System Design Components* content since its creation.

Date	Version	Changes
2016.10.31	16.1.0	• Implemented Intel rebranding. • Implemented Qsys Pro rebranding.
2016.05.03	16.0.0	Updated Address Span Extender • Address Span Extender register mapping better explained • Address Span Extender Parameters table added • Address Span Extender example added
2015.11.02	15.1.0	Changed instances of *Quartus II* to *Quartus Prime*.
2015.05.04	15.0.0	Avalon-MM Unaligned Burst Expansion Bridge and Avalon-MM Pipeline Bridge, **Maximum pending read transactions** parameter. Extended description.
December 2014	14.1.0	• AXI Timout Bridge. • Added notes to *Avalon-MM Clock Crossing Bridge* pertaining to: — SDC constraints for its internal asynchronous FIFOs. — FIFO-based clock crossing.
June 2014	14.0.0	• AXI Bridge support. • Address Span Extender updates. • Avalon-MM Unaligned Burst Expansion Bridge support.
November 2013	13.1.0	• Address Span Extender
May 2013	13.0.0	• Added Streaming Pipeline Stage support. • Added AMBA APB support.
November 2012	12.1.0	• Moved relevant content from the *Embedded Peripherals IP User Guide*.

Related Links

Altera Documentation Archive

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

15 Managing Metastability with the Quartus Prime Software

You can use the Quartus Prime software to analyze the average mean time between failures (MTBF) due to metastability caused by synchronization of asynchronous signals, and optimize the design to improve the metastability MTBF.

All registers in digital devices, such as FPGAs, have defined signal-timing requirements that allow each register to correctly capture data at its input ports and produce an output signal. To ensure reliable operation, the input to a register must be stable for a minimum amount of time before the clock edge (register setup time or t_{SU}) and a minimum amount of time after the clock edge (register hold time or t_H). The register output is available after a specified clock-to-output delay (t_{CO}).

If the data violates the setup or hold time requirements, the output of the register might go into a metastable state. In a metastable state, the voltage at the register output hovers at a value between the high and low states, which means the output transition to a defined high or low state is delayed beyond the specified t_{CO}. Different destination registers might capture different values for the metastable signal, which can cause the system to fail.

In synchronous systems, the input signals must always meet the register timing requirements, so that metastability does not occur. Metastability problems commonly occur when a signal is transferred between circuitry in unrelated or asynchronous clock domains, because the signal can arrive at any time relative to the destination clock.

The MTBF due to metastability is an estimate of the average time between instances when metastability could cause a design failure. A high MTBF (such as hundreds or thousands of years between metastability failures) indicates a more robust design. You should determine an acceptable target MTBF in the context of your entire system and taking in account that MTBF calculations are statistical estimates.

The metastability MTBF for a specific signal transfer, or all the transfers in a design, can be calculated using information about the design and the device characteristics. Improving the metastability MTBF for your design reduces the chance that signal transfers could cause metastability problems in your device.

The Quartus Prime software provides analysis, optimization, and reporting features to help manage metastability in Intel designs. These metastability features are supported only for designs constrained with the Quartus Prime Timing Analyzer. Both typical and worst-case MBTF values are generated for select device families.

Related Links

- Understanding Metastability in FPGAs
  For more information about metastability due to signal synchronization, its effects in FPGAs, and how MTBF is calculated
- Reliability Report

For information about Intel device reliability

15.1 Metastability Analysis in the Quartus Prime Software

When a signal transfers between circuitry in unrelated or asynchronous clock domains, the first register in the new clock domain acts as a synchronization register.

To minimize the failures due to metastability in asynchronous signal transfers, circuit designers typically use a sequence of registers (a synchronization register chain or synchronizer) in the destination clock domain to resynchronize the signal to the new clock domain and allow additional time for a potentially metastable signal to resolve to a known value. Designers commonly use two registers to synchronize a new signal, but a standard of three registers provides better metastability protection.

The timing analyzer can analyze and report the MTBF for each identified synchronizer that meets its timing requirements, and can generate an estimate of the overall design MTBF. The software uses this information to optimize the design MTBF, and you can use this information to determine whether your design requires longer synchronizer chains.

Related Links

- Metastability and MTBF Reporting on page 956
- MTBF Optimization on page 959
  In addition to reporting synchronization register chains and MTBF values found in the design, the Quartus Prime software can also protect these registers from optimizations that might negatively impact MTBF and can optimize the register placement and routing if the MTBF is too low.

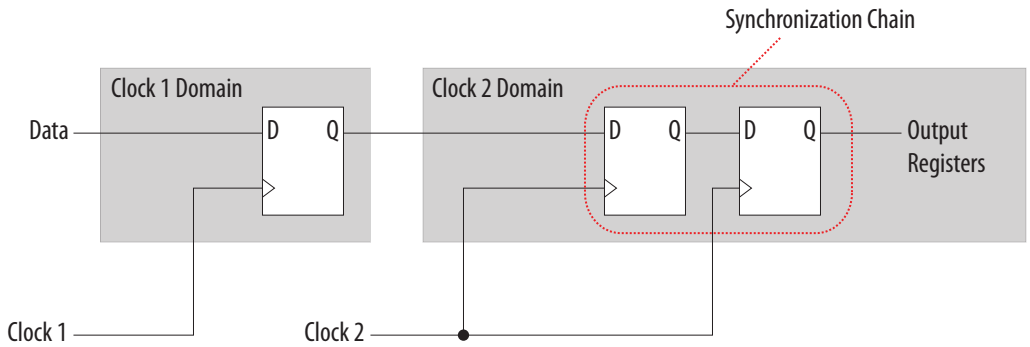
15.1.1 Synchronization Register Chains

A synchronization register chain, or synchronizer, is defined as a sequence of registers that meets the following requirements:

- The registers in the chain are all clocked by the same clock or phase-related clocks.
- The first register in the chain is driven asynchronously or from an unrelated clock domain.
- Each register fans out to only one register, except the last register in the chain.

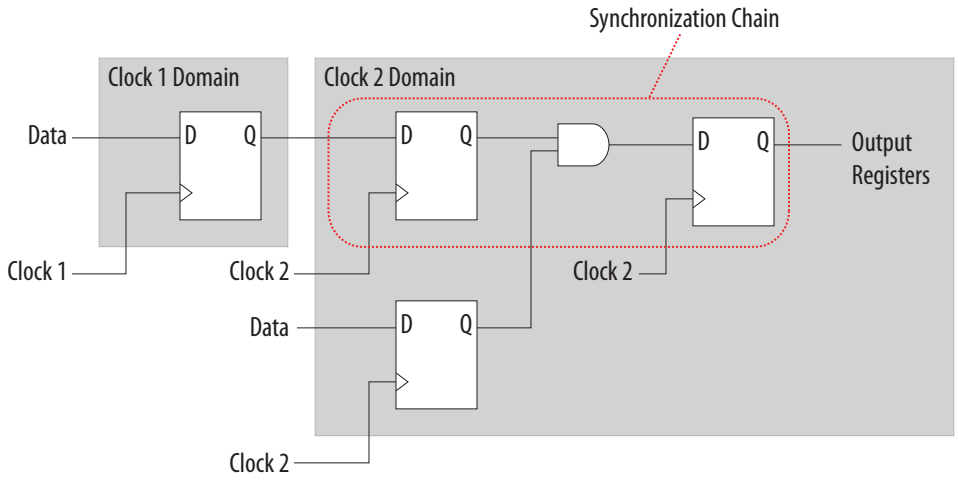
The length of the synchronization register chain is the number of registers in the synchronizing clock domain that meet the above requirements. The figure shows a sample two-register synchronization chain.

Figure 283. Sample Synchronization Register Chain



The path between synchronization registers can contain combinational logic as long as all registers of the synchronization register chain are in the same clock domain. The figure shows an example of a synchronization register chain that includes logic between the registers.

Figure 284. Sample Synchronization Register Chain Containing Logic



The timing slack available in the register-to-register paths of the synchronizer allows a metastable signal to settle, and is referred to as the available settling time. The available settling time in the MTBF calculation for a synchronizer is the sum of the output timing slacks for each register in the chain. Adding available settling time with additional synchronization registers improves the metastability MTBF.

Related Links

How Timing Constraints Affect Synchronizer Identification and Metastability Analysis on page 955

The timing analyzer can analyze metastability MTBF only if a synchronization chain meets its timing requirements.

15.1.2 Identifying Synchronizers for Metastability Analysis

The first step in enabling metastability MTBF analysis and optimization in the Quartus Prime software is to identify which registers are part of a synchronization register chain. You can apply synchronizer identification settings globally to automatically list possible synchronizers with the **Synchronizer identification** option on the **Timing Analyzer** page in the **Settings** dialog box.

Synchronization chains are already identified within most Intel FPGA intellectual property (IP) cores.

Related Links

Identifying Synchronizers for Metastability Analysis on page 955
The first step in enabling metastability MTBF analysis and optimization in the Quartus Prime software is to identify which registers are part of a synchronization register chain.

15.1.3 How Timing Constraints Affect Synchronizer Identification and Metastability Analysis

The timing analyzer can analyze metastability MTBF only if a synchronization chain meets its timing requirements. The metastability failure rate depends on the timing slack available in the synchronizer's register-to-register connections, because that slack is the available settling time for a potential metastable signal. Therefore, you must ensure that your design is correctly constrained with the real application frequency requirements to get an accurate MTBF report.

In addition, the **Auto** and **Forced If Asynchronous** synchronizer identification options use timing constraints to automatically detect the synchronizer chains in the design. These options check for signal transfers between circuitry in unrelated or asynchronous clock domains, so clock domains must be related correctly with timing constraints.

The timing analyzer views input ports as asynchronous signals unless they are associated correctly with a clock domain. If an input port fans out to registers that are not acting as synchronization registers, apply a `set_input_delay` constraint to the input port; otherwise, the input register might be reported as a synchronization register. Constraining a synchronous input port with a `set_max_delay` constraint for a setup (t_{SU}) requirement does not prevent synchronizer identification because the constraint does not associate the input port with a clock domain.

Instead, use the following command to specify an input setup requirement associated with a clock:

`set_input_delay -max -clock` *<clock name> <latch – launch – tsu requirement> <input port name>*

Registers that are at the end of false paths are also considered synchronization registers because false paths are not timing-analyzed. Because there are no timing requirements for these paths, the signal may change at any point, which may violate the t_{SU} and t_H of the register. Therefore, these registers are identified as synchronization registers. If these registers are not used for synchronization, you can turn off synchronizer identification and analysis. To do so, set **Synchronizer Identification** to **Off** for the first synchronization register in these register chains.

15.2 Metastability and MTBF Reporting

The Quartus Prime software reports the metastability analysis results in the Compilation Report and Timing Analyzer reports.

The MTBF calculation uses timing and structural information about the design, silicon characteristics, and operating conditions, along with the data toggle rate.

If you change the **Synchronizer Identification** settings, you can generate new metastability reports by rerunning the timing analyzer. However, you should rerun the Fitter first so that the registers identified with the new setting can be optimized for metastability MTBF.

Related Links

- Metastability Reports on page 956
  Metastability reports provide summaries of the metastability analysis results. In addition to the MTBF Summary and Synchronizer Summary reports, the Timing Analyzer tool reports additional statistics in a report for each synchronizer chain.
- MTBF Optimization on page 959
  In addition to reporting synchronization register chains and MTBF values found in the design, the Quartus Prime software can also protect these registers from optimizations that might negatively impact MTBF and can optimize the register placement and routing if the MTBF is too low.
- Synchronizer Data Toggle Rate in MTBF Calculation on page 958
  The MTBF calculations assume the data being synchronized is switching at a toggle rate of 12.5% of the source clock frequency. That is, the arriving data is assumed to switch once every eight source clock cycles.
- Understanding Metastability in FPGAs
  For more information about how metastability MTBF is calculated

15.2.1 Metastability Reports

Metastability reports provide summaries of the metastability analysis results. In addition to the MTBF Summary and Synchronizer Summary reports, the Timing Analyzer tool reports additional statistics in a report for each synchronizer chain.

Note: If the design uses only the **Auto Synchronizer Identification** setting, the reports list likely synchronizers but do not report MTBF. To obtain an MTBF for each register chain, force identification of synchronization registers.

Note: If the synchronizer chain does not meet its timing requirements, the reports list identified synchronizers but do not report MTBF. To obtain MTBF calculations, ensure that the design is properly constrained and that the synchronizer meets its timing requirements.

Related Links

- Identifying Synchronizers for Metastability Analysis on page 955
  The first step in enabling metastability MTBF analysis and optimization in the Quartus Prime software is to identify which registers are part of a synchronization register chain.
- How Timing Constraints Affect Synchronizer Identification and Metastability Analysis on page 955

The timing analyzer can analyze metastability MTBF only if a synchronization chain meets its timing requirements.

15.2.1.1 MTBF Summary Report

The MTBF Summary reports an estimate of the overall robustness of cross-clock domain and asynchronous transfers in the design. This estimate uses the MTBF results of all synchronization chains in the design to calculate an MTBF for the entire design.

15.2.1.1.1 Typical and Worst-Case MTBF of Design

The MTBF Summary Report shows the **Typical MTBF of Design** and the **Worst-Case MTBF of Design** for supported fully-characterized devices. The typical MTBF result assumes typical conditions, defined as nominal silicon characteristics for the selected device speed grade, as well as nominal operating conditions. The worst case MTBF result uses the worst case silicon characteristics for the selected device speed grade.

When you analyze multiple timing corners in the timing analyzer, the MTBF calculation may vary because of changes in the operating conditions, and the timing slack or available metastability settling time. Intel recommends running multi-corner timing analysis to ensure that you analyze the worst MTBF results, because the worst timing corner for MTBF does not necessarily match the worst corner for timing performance.

Related Links

Timing Analyzer page

15.2.1.1.2 Synchronizer Chains

The MTBF Summary report also lists the **Number of Synchronizer Chains Found** and the length of the **Shortest Synchronizer Chain**, which can help you identify whether the report is based on accurate information.

If the number of synchronizer chains found is different from what you expect, or if the length of the shortest synchronizer chain is less than you expect, you might have to add or change **Synchronizer Identification** settings for the design. The report also provides the **Worst Case Available Settling Time**, defined as the available settling time for the synchronizer with the worst MTBF.

You can use the reported **Fraction of Chains for which MTBFs Could Not be Calculated** to determine whether a high proportion of chains are missing in the metastability analysis. A fraction of 1, for example, means that MTBF could not be calculated for any chains in the design. MTBF is not calculated if you have not identified the chain with the appropriate **Synchronizer identification** option, or if paths are not timing-analyzed and therefore have no valid slack for metastability analysis. You might have to correct your timing constraints to enable complete analysis of the applicable register chains.

15.2.1.1.3 Increasing Available Settling Time

The MTBF Summary report specifies how an increase of 100ps in available settling time increases the MTBF values. If your MTBF is not satisfactory, this metric can help you determine how much extra slack would be required in your synchronizer chain to allow you to reach the desired design MTBF.

15.2.1.2 Synchronizer Summary Report

The **Synchronizer Summary** lists the synchronization register chains detected in the design depending on the Synchronizer Identification setting.

The **Source Node** is the register or input port that is the source of the asynchronous transfer. The **Synchronization Node** is the first register of the synchronization chain. The **Source Clock** is the clock domain of the source node, and the **Synchronization Clock** is the clock domain of the synchronizer chain.

This summary reports the calculated **Worst-Case MTBF**, if available, and the **Typical MTBF**, for each appropriately identified synchronization register chain that meets its timing requirement.

Related Links

Synchronizer Chain Statistics Report in the Timing Analyzer on page 958
The timing analyzer provides an additional report for each synchronizer chain.

15.2.1.3 Synchronizer Chain Statistics Report in the Timing Analyzer

The timing analyzer provides an additional report for each synchronizer chain.

The **Chain Summary** tab matches the Synchronizer Summary information described in "C**Synchronizer Summary Report", while the **Statistics** tab adds more details, including whether the **Method of Synchronizer Identification** was **User Specified** (with the **Forced if Asynchronous** or **Forced** settings for the **Synchronizer Identification** setting), or **Automatic** (with the **Auto** setting). The **Number of Synchronization Registers in Chain** report provides information about the parameters that affect the MTBF calculation, including the **Available Settling Time** for the chain and the **Data Toggle Rate Used in MTBF Calculation**.

The following information is also included to help you locate the chain in your design:

- **Source Clock** and **Asynchronous Source** node of the signal.
- **Synchronization Clock** in the destination clock domain.
- Node names of the **Synchronization Registers** in the chain.

Related Links

Synchronizer Data Toggle Rate in MTBF Calculation on page 958
The MTBF calculations assume the data being synchronized is switching at a toggle rate of 12.5% of the source clock frequency. That is, the arriving data is assumed to switch once every eight source clock cycles.

15.2.2 Synchronizer Data Toggle Rate in MTBF Calculation

The MTBF calculations assume the data being synchronized is switching at a toggle rate of 12.5% of the source clock frequency. That is, the arriving data is assumed to switch once every eight source clock cycles.

If multiple clocks apply, the highest frequency is used. If no source clocks can be determined, the data rate is taken as 12.5% of the synchronization clock frequency.

If you know an approximate rate at which the data changes, specify it with the **Synchronizer Toggle Rate** assignment in the Assignment Editor. You can also apply this assignment to an entity or the entire design. Set the data toggle rate, in number

of transitions per second, on the first register of a synchronization chain. The timing analyzer takes the specified rate into account when computing the MTBF of that particular register chain. If a data signal never toggles and does not affect the reliability of the design, you can set the **Synchronizer Toggle Rate** to **0** for the synchronization chain so the MTBF is not reported. To apply the assignment with Tcl, use the following command:

```
set_instance_assignment -name SYNCHRONIZER_TOGGLE_RATE <toggle rate in
transitions/second> -to <register name>
```

In addition to **Synchronizer Toggle Rate,** there are two other assignments associated with toggle rates, which are not used for metastability MTBF calculations. The I/O Maximum Toggle Rate is only used for pins, and specifies the worst-case toggle rates used for signal integrity purposes. The Power Toggle Rate assignment is used to specify the expected time-averaged toggle rate, and is used by the Power Analyzer to estimate time-averaged power consumption.

15.3 MTBF Optimization

In addition to reporting synchronization register chains and MTBF values found in the design, the Quartus Prime software can also protect these registers from optimizations that might negatively impact MTBF and can optimize the register placement and routing if the MTBF is too low.

Synchronization register chains must first be explicitly identified as synchronizers. Intel recommends that you set **Synchronizer Identification** to **Forced If Asynchronous** for all registers that are part of a synchronizer chain.

Optimization algorithms, such as register duplication and logic retiming in physical synthesis, are not performed on identified synchronization registers. The Fitter protects the number of synchronization registers specified by the **Synchronizer Register Chain Length** option.

In addition, the Fitter optimizes identified synchronizers for improved MTBF by placing and routing the registers to increase their output setup slack values. Adding slack in the synchronizer chain increases the available settling time for a potentially metastable signal, which improves the chance that the signal resolves to a known value, and exponentially increases the design MTBF. The Fitter optimizes the number of synchronization registers specified by the **Synchronizer Register Chain Length** option.

Metastability optimization is **on** by default. To view or change the **Optimize Design for Metastability** option, click **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Fitter)**. To turn the optimization on or off with Tcl, use the following command:

```
set_global_assignment -name OPTIMIZE_FOR_METASTABILITY <ON|OFF>
```

Related Links

Identifying Synchronizers for Metastability Analysis on page 955
The first step in enabling metastability MTBF analysis and optimization in the Quartus Prime software is to identify which registers are part of a synchronization register chain.

15.3.1 Synchronization Register Chain Length

The **Synchronization Register Chain Length** option specifies how many registers should be protected from optimizations that might reduce MTBF for each register chain, and controls how many registers should be optimized to increase MTBF with the **Optimize Design for Metastability** option.

For example, if the **Synchronization Register Chain Length** option is set to **2**, optimizations such as register duplication or logic retiming are prevented from being performed on the first two registers in all identified synchronization chains. The first two registers are also optimized to improve MTBF when the **Optimize Design for Metastability** option is turned on.

The default setting for the **Synchronization Register Chain Length** option is **2**. The first register of a synchronization chain is always protected from operations that might reduce MTBF, but you should set the protection length to protect more of the synchronizer chain. Intel recommends that you set this option to the maximum length of synchronization chains you have in your design so that all synchronization registers are preserved and optimized for MTBF.

Click **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Synthesis)** to change the global **Synchronization Register Chain Length** option.

You can also set the **Synchronization Register Chain Length** on a node or an entity in the Assignment Editor. You can set this value on the first register in a synchronization chain to specify how many registers to protect and optimize in this chain. This individual setting is useful if you want to protect and optimize extra registers that you have created in a specific synchronization chain that has low MTBF, or optimize less registers for MTBF in a specific chain where the maximum frequency or timing performance is not being met. To make the global setting with Tcl, use the following command:

```
set_global_assignment -name SYNCHRONIZATION_REGISTER_CHAIN_LENGTH <number of
registers>
```

To apply the assignment to a design instance or the first register in a specific chain with Tcl, use the following command:

```
set_instance_assignment -name SYNCHRONIZATION_REGISTER_CHAIN_LENGTH <number of
registers> -to <register or instance name>
```

15.4 Reducing Metastability Effects

You can check your design's metastability MTBF in the Metastability Summary report, and determine an acceptable target MTBF given the context of your entire system and the fact that MTBF calculations are statistical estimates. A high metastability MTBF (such as hundreds or thousands of years between metastability failures) indicates a more robust design.

This section provides guidelines to ensure complete and accurate metastability analysis, and some suggestions to follow if the Quartus Prime metastability reports calculate an unacceptable MTBF value. The Timing Optimization Advisor (available from the Tools menu) gives similar suggestions in the Metastability Optimization section.

Related Links

Metastability Reports on page 956
Metastability reports provide summaries of the metastability analysis results. In addition to the MTBF Summary and Synchronizer Summary reports, the Timing Analyzer tool reports additional statistics in a report for each synchronizer chain.

15.4.1 Apply Complete System-Centric Timing Constraints for the Timing Analyzer

To enable the Quartus Prime metastability features, make sure that the timing analyzer is used for timing analysis.

Ensure that the design is fully timing constrained and that it meets its timing requirements. If the synchronization chain does not meet its timing requirements, MTBF cannot be calculated. If the clock domain constraints are set up incorrectly, the signal transfers between circuitry in unrelated or asynchronous clock domains might be identified incorrectly.

Use industry-standard system-centric I/O timing constraints instead of using FPGA-centric timing constraints.

You should use `set_input_delay` constraints in place of `set_max_delay` constraints to associate each input port with a clock domain to help eliminate false positives during synchronization register identification.

Related Links

How Timing Constraints Affect Synchronizer Identification and Metastability Analysis on page 955
The timing analyzer can analyze metastability MTBF only if a synchronization chain meets its timing requirements.

15.4.2 Force the Identification of Synchronization Registers

Use the guidelines in *"Identifying Synchronizers for Metastability Analysis"* to ensure the software reports and optimizes the appropriate register chains.

Identify synchronization registers with the **Synchronizer Identification** set to **Forced If Asynchronous** in the Assignment Editor. If there are any registers that the software detects as synchronous but you want to be analyzed for metastability, apply the **Forced** setting to the first synchronizing register. Set **Synchronizer Identification** to **Off** for registers that are not synchronizers for asynchronous signals or unrelated clock domains.

To help you find the synchronizers in your design, you can set the global **Synchronizer Identification** setting on the **Timing Analyzer** page of the **Settings** dialog box to **Auto** to generate a list of all the possible synchronization chains in your design.

Related Links

Identifying Synchronizers for Metastability Analysis on page 955
The first step in enabling metastability MTBF analysis and optimization in the Quartus Prime software is to identify which registers are part of a synchronization register chain.

15.4.3 Set the Synchronizer Data Toggle Rate

The MTBF calculations assume the data being synchronized is switching at a toggle rate of 12.5% of the source clock frequency.

To obtain a more accurate MTBF for a specific chain or all chains in your design, set the **Synchronizer Toggle Rate**.

Related Links

Synchronizer Data Toggle Rate in MTBF Calculation on page 958
The MTBF calculations assume the data being synchronized is switching at a toggle rate of 12.5% of the source clock frequency. That is, the arriving data is assumed to switch once every eight source clock cycles.

15.4.4 Optimize Metastability During Fitting

Ensure that the **Optimize Design for Metastability** setting is turned on.

Related Links

MTBF Optimization on page 959
In addition to reporting synchronization register chains and MTBF values found in the design, the Quartus Prime software can also protect these registers from optimizations that might negatively impact MTBF and can optimize the register placement and routing if the MTBF is too low.

15.4.5 Increase the Length of Synchronizers to Protect and Optimize

Increase the Synchronizer Chain Length parameter to the maximum length of synchronization chains in your design. If you have synchronization chains longer than 2 identified in your design, you can protect the entire synchronization chain from operations that might reduce MTBF and allow metastability optimizations to improve the MTBF.

Related Links

Synchronization Register Chain Length on page 960
The **Synchronization Register Chain Length** option specifies how many registers should be protected from optimizations that might reduce MTBF for each register chain, and controls how many registers should be optimized to increase MTBF with the **Optimize Design for Metastability** option.

15.4.6 Increase the Number of Stages Used in Synchronizers

Designers commonly use two registers in a synchronization chain to minimize the occurrence of metastable events, and a standard of three registers provides better metastability protection. However, synchronization chains with two or even three registers may not be enough to produce a high enough MTBF when the design runs at high clock and data frequencies.

If a synchronization chain is reported to have a low MTBF, consider adding an additional register stage to your synchronization chain. This additional stage increases the settling time of the synchronization chain, allowing more opportunity for the signal to resolve to a known state during a metastable event. Additional settling time increases the MTBF of the chain and improves the robustness of your design. However, adding a synchronization stage introduces an additional stage of latency on the signal.

If you use the Altera FIFO IP core with separate read and write clocks to cross clock domains, increase the metastability protection (and latency) for better MTBF. In the DCFIFO parameter editor, choose the **Best metastability protection, best fmax, unsynchronized clocks** option to add three or more synchronization stages. You can increase the number of stages to more than three using the **How many sync stages?** setting.

15.4.7 Select a Faster Speed Grade Device

The design MTBF depends on process parameters of the device used. Faster devices are less susceptible to metastability issues. If the design MTBF falls significantly below the target MTBF, switching to a faster speed grade can improve the MTBF substantially.

15.5 Scripting Support

You can run procedures and make settings described in this chapter in a Tcl script. You can also run some procedures at a command prompt. For detailed information about scripting command options, refer to the Quartus Prime Command-Line and Tcl API Help browser.

To run the Help browser, type the following command at the command prompt:

```
quartus_sh --qhelp r
```

Related Links

- Tcl Scripting
  For more information about Tcl scripting
- Quartus Prime Settings File Reference Manual
  For more information about settings and constraints in the Quartus Prime software
- Command-Line Scripting
  For more information about command-line scripting
- About Quartus Prime Scripting
  For more information about command-line scripting

15.5.1 Identifying Synchronizers for Metastability Analysis

To apply the global Synchronizer Identification assignment, use the following command:

```
set_global_assignment -name SYNCHRONIZER_IDENTIFICATION <OFF|AUTO|"FORCED IF
ASYNCHRONOUS">
```

To apply the **Synchronizer Identification** assignment to a specific register or instance, use the following command:

```
set_instance_assignment -name SYNCHRONIZER_IDENTIFICATION <AUTO|"FORCED IF
ASYNCHRONOUS"|FORCED|OFF> -to <register or instance name>
```

15.5.2 Synchronizer Data Toggle Rate in MTBF Calculation

To specify a toggle rate for MTBF calculations as described on page "R**Synchronizer Data Toggle Rate in MTBF Calculation", use the following command:

```
set_instance_assignment -name SYNCHRONIZER_TOGGLE_RATE <toggle rate in
transitions/second> -to <register name>
```

Related Links

Synchronizer Data Toggle Rate in MTBF Calculation on page 958
The MTBF calculations assume the data being synchronized is switching at a toggle rate of 12.5% of the source clock frequency. That is, the arriving data is assumed to switch once every eight source clock cycles.

15.5.3 report_metastability and Tcl Command

If you use a command-line or scripting flow, you can generate the metastability analysis reports described in "C**Metastability Reports" outside of the Quartus Prime and user interfaces.

The table describes the options for the `report_metastability` and Tcl command.

Table 255. report_metastabilty Command Options

Option	Description
`-append`	If output is sent to a file, this option appends the result to that file. Otherwise, the file is overwritten.
`-file <name>`	Sends the results to an ASCII or HTML file. The extension specified in the file name determines the file type — either **`*.txt`** or **`*.html`**.
`-panel_name <name>`	Sends the results to the panel and specifies the name of the new panel.
`-stdout`	Indicates the report be sent to the standard output, via messages. This option is required only if you have selected another output format, such as a file, and would also like to receive messages.

Related Links

Metastability Reports on page 956
Metastability reports provide summaries of the metastability analysis results. In addition to the MTBF Summary and Synchronizer Summary reports, the Timing Analyzer tool reports additional statistics in a report for each synchronizer chain.

15.5.4 MTBF Optimization

To ensure that metastability optimization described on page "C**MTBF Optimization" is turned on (or to turn it off), use the following command:

```
set_global_assignment -name OPTIMIZE_FOR_METASTABILITY <ON|OFF>
```

Related Links

MTBF Optimization on page 959

In addition to reporting synchronization register chains and MTBF values found in the design, the Quartus Prime software can also protect these registers from optimizations that might negatively impact MTBF and can optimize the register placement and routing if the MTBF is too low.

15.5.5 Synchronization Register Chain Length

To globally set the number of registers in a synchronization chain to be protected and optimized as described on page "C**Synchronization Register Chain Length", use the following command:

```
set_global_assignment -name SYNCHRONIZATION_REGISTER_CHAIN_LENGTH <number of registers>
```

To apply the assignment to a design instance or the first register in a specific chain, use the following command:

```
set_instance_assignment -name SYNCHRONIZATION_REGISTER_CHAIN_LENGTH <number of registers> -to <register or instance name>
```

Related Links

Synchronization Register Chain Length on page 960
The **Synchronization Register Chain Length** option specifies how many registers should be protected from optimizations that might reduce MTBF for each register chain, and controls how many registers should be optimized to increase MTBF with the **Optimize Design for Metastability** option.

15.6 Managing Metastability

Intel's Quartus Prime software provides industry-leading analysis and optimization features to help you manage metastability in your FPGA designs. Set up your Quartus Prime project with the appropriate constraints and settings to enable the software to analyze, report, and optimize the design MTBF. Take advantage of these features in the Quartus Prime software to make your design more robust with respect to metastability.

15.7 Document Revision History

Table 256. Document Revision History

Date	Version	Changes
2016.10.31	16.1.0	• Implemented Intel rebranding.
2015.11.02	15.1.0	• Changed instances of *Quartus II* to *Quartus Prime*.
2014.12.15	14.1.0	Updated location of Fitter Settings, Analysis & Synthesis Settings, and Physical Optimization Settings to Compiler Settings.
June 2014	14.0.0	Updated formatting.
June 2012	12.0.0	Removed survey link.
November 2011	10.0.2	Template update.
December 2010	10.0.1	Changed to new document template.

continued...

Date	Version	Changes
July 2010	10.0.0	Technical edit.
November 2009	9.1.0	Clarified description of synchronizer identification settings. Minor changes to text and figures throughout document.
March 2009	9.0.0	Initial release.

Related Links

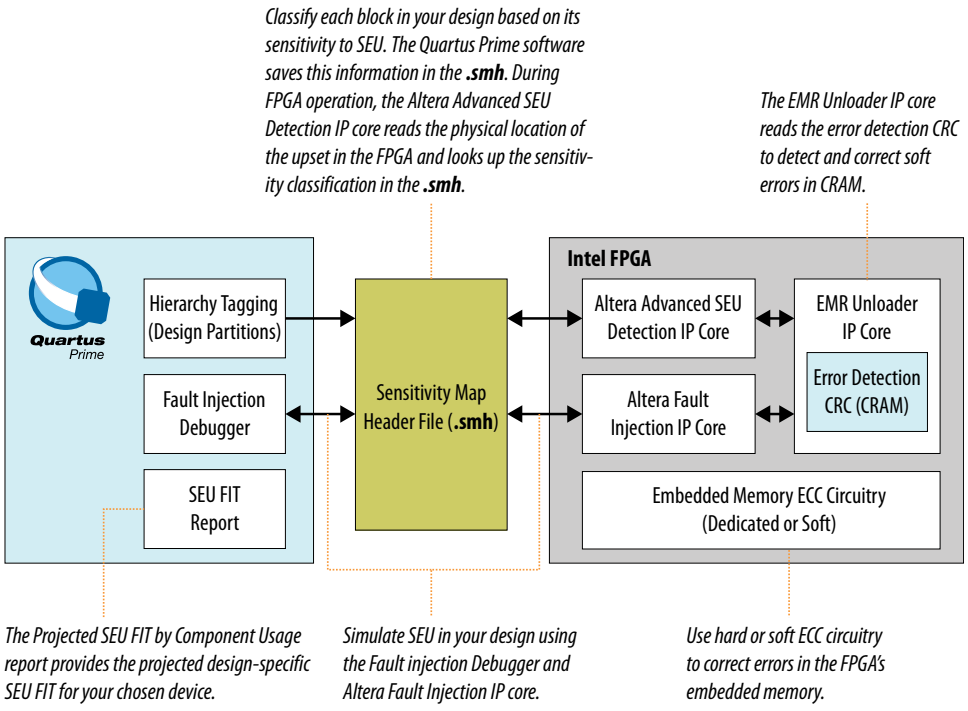
Altera Documentation Archive

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

16 Mitigating Single Event Upset

Single event upsets (SEUs) are rare, unintended changes in the state of an FPGA's internal memory elements caused by cosmic radiation effects. The change in state is a soft error and the FPGA incurs no permanent damage. Because of the unintended memory state, the FPGA may operate erroneously until background scrubbing fixes the upset. The Quartus Prime software offers several features to detect and correct the effects of SEU, or soft errors, as well as to characterize the effects of SEU on your designs. Additionally, some Intel FPGAs contain dedicated circuitry to help detect and correct errors.

Figure 285. Tools, IP, and Circuitry for Detecting and Correcting SEU



Intel FPGAs have memory in user logic (block memory and registers) and in Configuration Random Access Memory (CRAM). The Quartus Prime Programmer loads the CRAM with a **.sof** file. Then, the CRAM configures all FPGA logic and routing. If an SEU strikes a CRAM bit, the effect can be harmless if the device does not use the CRAM bit. However, the affect can be severe if the SEU affects critical logic or internal signal routing (such as a lookup table bit).

Often, a design does not require SEU mitigation because of the low chance of occurrence. However, for highly complex systems, such as systems with multiple high-density components, the error rate may be a significant system design factor. If your

system includes multiple FPGAs and requires very high reliability and availability, you should consider the implications of soft errors. Use the techniques in this chapter to detect and recover from these types of errors.

Related Links

- Introduction to Single Event Upsets
- Understanding Single Event Functional Interrupts in FPGA Designs White Paper

16.1 Understanding Failure Rates

One can express the Soft Error Rate (SER) or SEU reliability as Failure-in-Time (FIT) units, defined as one soft error occurrence every billion hours of operation. A design that has 5,000 FIT experiences a mean of 5,000 SEU events in 1 billion hours (or 8,333.33 years). Because SEU events are statistically independent, FIT is additive: if a single FPGA has 5,000 FIT, then 10 FPGAs have 50,000 FIT (or 50K failures in 8,333 years).

Alternatively, one can measure reliability by the mean time to failure (MTTF), which is the reciprocal of the FIT or 1/FIT. For a FIT of 5,000 in standard units of failures/billion hours, MTTF is:

1 /(5,000/1Bh) =1 billion/5,000 = 200,000 hours = 22.83 years

SEU events follow a Poisson distribution and the cumulative distribution function (CDF) for mean time between failures (MTBF) is an exponential distribution.

Neutron SEU incidence varies by altitude, latitude, and other environmental factors. The Quartus Prime software provides SEU FIT reports based on compiles for sea level in Manhattan, New York. The JESD 89A specification defines the test parameters. You can convert the data to other locations and altitudes using calculators, such as those at www.seutest.com. Additionally, you can include the relative neutron flux (calculated at www.seutest.com) in your project's Quartus Prime Settings File (**.qsf**) to adjust the SEU rates.

Related Links

- Poisson Distribution
- Exponential Distribution
- Soft-error Testing Resources
- JEDEC Standard 89A
- Understanding the SEU FIT Reports on page 973
  Intel's device Reliability Report shows raw SEU test data. This data is the FIT rate that the design would have if it uses every configuration RAM bit, M20K bit, and every flipflop in the chip.

16.2 Mitigating SEU Effects in Embedded User RAM

Stratix V, Stratix IV, Stratix III, and Arria V GZ FPGAs offer dedicated error correcting code (ECC) circuitry for embedded memory blocks. FPGA families that do not have dedicated ECC circuitry support ECC by implementing a soft IP core. You can reduce the FIT rate for these memories to near zero by enabling the ECC encode/decode

blocks. On ingress, the ECC encoder adds 8 bits of redundancy to a 32 bit word. On egress, the decoder converts the 40 bit word back to 32 bits. You use the redundant bits to detect and correct errors in the data resulting from SEU.

The existence of hard ECC and the strength of the ECC code (number of corrected and detected bits) varies by device family. Refer to the device handbook for details. If a device does not have a hard ECC block you can add ECC parity or use an ECC IP core.

The SRAM memories associated with processor subsystems, such as for SoC devices, contain dedicated hard ECC. You do not need to take action to protect these memories.

For more information on embedded memories and ECC, refer to the *Embedded Memory (RAM: 1-PORT, RAM:2-PORT, ROM: 1-PORT, and ROM: 2-PORT) User Guide*.

Related Links

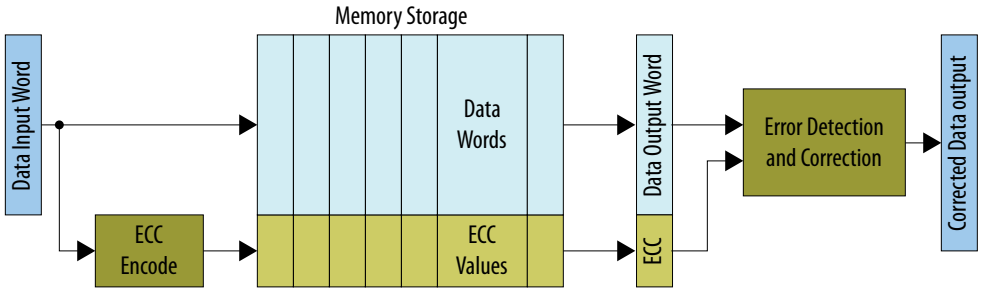
Embedded Memory (RAM: 1-PORT, RAM:2-PORT, ROM: 1-PORT, and ROM: 2-PORT) User Guide

16.2.1 Configuring RAM to Enable ECC

To enable ECC, configure the RAM as a 2-port RAM with independent read and write addresses. Using this feature does not reduce the available logic.

Although the ECC checking function results in some additional output delay, the hard ECC has a much higher $\mathbf{f_{MAX}}$ compared with an equivalent soft ECC block implemented in general logic. Additionally, you can pipeline the hard IP in the M20K block by configuring the ECC-enabled RAM to use an output register at the corrected data output port. This implementation increases performance and adds latency.
For devices without dedicated circuitry, you can implement ECC by instantiating the ALTECC IP core, which performs ECC generation and checking functions.

Figure 286. Memory Storage and ECC



Related Links

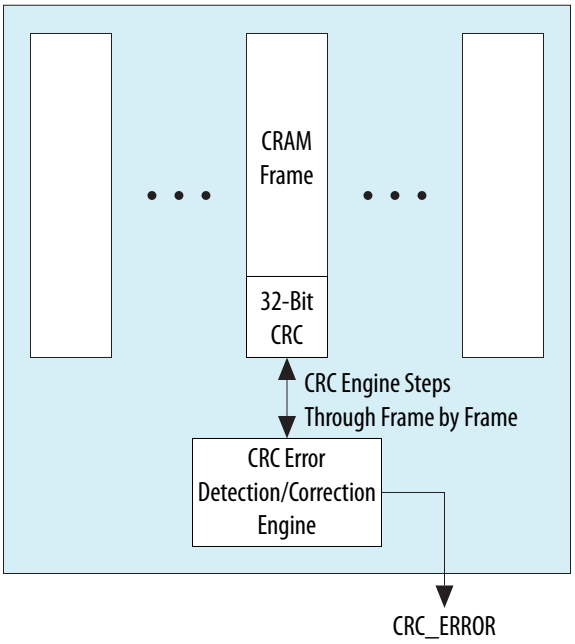
ALTECC (Error Correction Code: Encoder/Decoder)

16.3 Mitigating SEU Effects in Configuration RAM

Use error detect CRC (EDCRC) hard blocks to detect and correct soft errors in CRAM. These EDCRC blocks are similar to those that protect internal user memory.

The FPGA contains frames of CRAM. The size and number of frames is device specific. The device continually checks the CRAM frames for errors by loading each frame into a data register. The EDCRC block checks the frame for errors. When the FPGA finds a soft error, the FPGA asserts its CRC_ERROR pin. Monitor this pin in your system. When your system detects that the FPGA asserted this pin during operation, indicating the FPGA detected a soft error in the configuration RAM, the system can take action to recover from the error. For example, the system can perform a soft reset (after waiting for background scrubbing), reprogram the FPGA, or classify the error as benign and ignore it.

Figure 287. CRAM Frame



Related Links

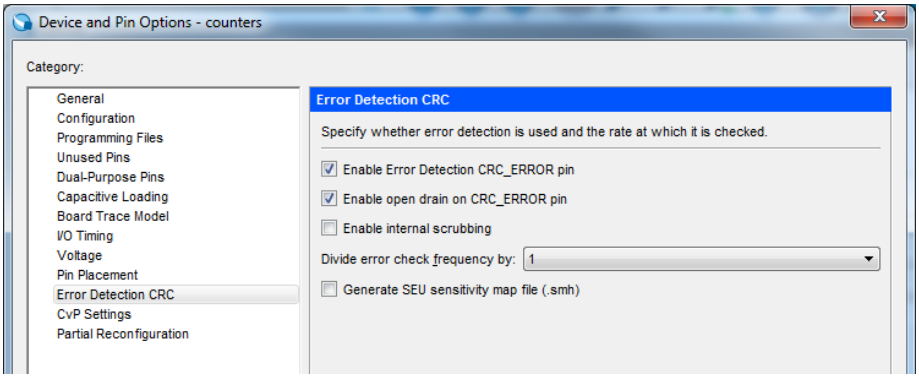
Single Event Upsets

16.3.1 Scanning CRAM Frames

Specify device and pin options to use CRAM frame error detection.

1. To enable CRAM frame scanning in the Quartus Prime software, click **Assignments ➤ Device ➤ Device and Pin Options ➤ Error Detection CRC** and turn on **Enable Error Detection CRC_ERROR pin**.

Figure 288. Enable Error Detection CRC_ERROR Pin



2. To enable the CRC_ERROR pin as an open-drain output, turn on **Enable open drain on CRC_ERROR pin**.
3. To guarantee the availability of a clock, the EDCRC function operates on an independent clock generated internally on the FPGA itself. To enable EDCRC operation on a divided version of the clock select a value from **Divide error check frequency by**.

16.4 Internal Scrubbing

Arria 10 support automatic CRAM error correction without reloading the original CRAM contents from an external copy of the original `.sof`.

The EDCRC calculates and stores redundancy fields of the FPGA's configuration bits. Therefore, the FPGA can correct errors automatically. This automatic correction is known as internal scrubbing. To enable internal scrubbing, click **Assignments ➤ Device ➤ Device and Pin Options ➤ Error Detection CRC** and turn on the **Enable internal scrubbing** option.

If the FPGA finds a CRC error in a CRAM frame, the FPGA reconstructs the frame from the error correcting code calculated for that frame. Then the FPGA writes the corrected frame into the CRAM.

Note: If you enable internal scrubbing, you must still plan a recovery sequence. Although scrubbing can restore the CRAM array to intended configuration, latency occurs between the soft error detection and correction. Because of the large number of configuration bits the FPGA must scan, this latency can be up to 100 milliseconds for large devices. Therefore, the FPGA may operate with errors during that period.

Related Links

Error Detection CRC Page

16.5 Recovering from SEU

After correcting a CRAM bit flip, the FPGA is in its original configuration with respect to logic and routing. However, the FPGA may have an illegal internal state. For example, the state may be invalid because SEUs corrupted the FPGA configuration during operation. Errors due to faulty operation can propagate elsewhere within the FPGA or

to the system outside the FPGA. When recovering from SEU, the system should reset the FPGA to a known state. During your design process, determine the possible SEU outcomes and design a recovery response.

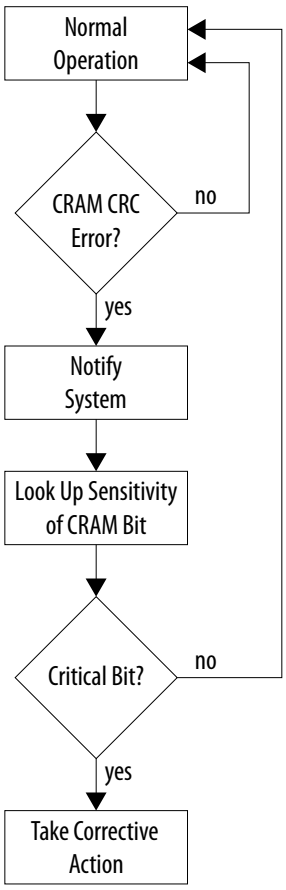
16.6 Planning for SEU Recovery

Reconfiguring a running FPGA typically has a significant system impact. When planning for SEU recovery, account for the time required to bring the FPGA to a state consistent with the current state of the system. For example, if an internal state machine is in an illegal state, it may require reset. Also, the surrounding logic may need to account for this unexpected operation.

Often an SEU impacts CRAM bits that the implemented design does not use (for example, CRAM bits that control unused control logic and routing wires). Depending on the implementation, the most heavily utilized FPGAs only use about 40% of available CRAM bits. Therefore, only 40% of potential SEU events in the entire FPGA require intervention, and you can ignore the remaining 60%. Designs that do not completely fill the FPGA use even fewer available CRAM bits.

You may determine that portions of the implemented design are not critical to the FPGA's function. Examples include test circuitry that is not important to the FPGA operation, or other non-critical functions that the system may log but does not need to reprogram or reset.

Figure 289. Sensitivity Processing Flow



The ratio of SEU strikes versus functional interrupts is the Single Event Functional Interrupt (SEFI) ratio. Minimizing this ratio improves SEU mitigation.

Related Links

- AN 737: SEU Detection and Recovery in Arria 10 Devices
- Understanding Single Event Functional Interrupts in FPGA Designs

16.7 Understanding the Quartus Prime SEU FIT Reports

Intel's device Reliability Report shows raw SEU test data. This data is the FIT rate that the design would have if it uses every configuration RAM bit, M20K bit, and every flipflop in the chip. Because real FPGA designs cannot have 100% bit utilization, the raw report is a pessimistic number. For example, even a 100% full design does not use 100% of its available routing, some LUTs are not full 6-input LUTs, and most designs do not use all available hard IP blocks.

The Projected SEU FIT by Component Usage report provides a design-specific SEU report for the device. The report shows all bits (the raw FIT), utilized bits (only resources the design actually uses), and the ECC-mitigated bits.

The SEU FIT Parameters report shows the environmental assumptions that influence the FIT/Mb values.

- **Altitude** represents the default altitude (above sea-level).
- **Neutron Flux Multiplier** is the relative flux for the default location, which is New York City per JESD specification. The default is 1. Change the setting by adding the following assignment to your `.qsf`:

  `set_global_assignment RELATIVE_NEUTRON_FLUX <relative_flux>`

  Note: You can compute scaled values using the JESD published equations for altitude, latitude, and longitude. Websites, such as www.seutest.com, can make this computation for you.
- **Alpha Flux** is the default for standard Intel packages; you cannot override the default.

  Note: When you change the relative **Neutron Flux Multiplier**, the Quartus Prime software only scales the neutron component of FIT. Alpha flux is not affected by location.

Figure 290. SEU FIT Parameters

SEU FIT Parameters

	Parameter	value
1	Device	5SGXEA7N2F45I2
2	Altitude	0.00
3	Neutron Flux	JESD - 89A assuming sea - level(> 10 MeV) 13.00 n / hr / cm2
4	Neutron Flux Multiplier	1.000
5	Alpha Flux	0.001 CPH/cm^2

Change the Neutron Flux Multiplier using the assignment: set_global_assignment RELATIVE_NEUTRON_FLUX <relative_flux>

Figure 291. Projected SEU FIT by Component Usage Report

The Projected SEU FIT by Component Usage report shows the different components (or cell types) that comprise the total FIT rate, and SEU FIT calculation results.

Projected SEU FIT by Component Usage

	Component	Raw	Utilized	w/ECC	AVF 0.5	AVF 0.25
1	⊟ Configuration (CRAM)	8486	3817	3817	1909	955
1	--Logic	2125	1071	1071	536	268
2	--Routing	6314	2716	2716	1358	679
3	--I/O config	47	30	30	15	8
2	⊟ RAM	41696	8593	1218	609	304
1	--HARD-IP (E.G. PCIe)	1446	692	0	0	0
2	--Embedded RAM	40250	7901	1218	609	304
3	--MLAB (LUTRAM)	0	0	0	0	0
3	⊟ Registers	2563	794	794	396	198
1	--Hard-IP FF	474	177	177	88	44
2	--DSP/M20K FF	298	61	61	30	15
3	--Design FF	1791	556	556	278	139
4						
5	TOTAL	52745	13204	5829	2914	1457

Note: Arria 10 FPGAs support the Projected SEU FIT by Component Usage report.

Related Links

Soft-error Testing Resources

16.7.1 Component FIT Rates

An Intel FPGA's sensitivity to soft errors varies by process technology, the component type, and your design choices when implementing the component (such as tradeoffs between area/delay and SEU rates).

The Projected SEU FIT by Component report shows FIT for the following components:

- SRAM embedded memory in embedded processors hard IP and M20K or M10K blocks
- CRAM used for LUT masks and routing configuration bits
- LABs in MLAB mode
- I/O configuration registers, which the FPGA implements differently than CRAM and design flipflops
- Standard flipflops the design uses in the address and data registers of M20K blocks, in DSP blocks, and in hard IP
- User flipflops the design implements in logic cells (ALMs or LEs)

16.7.2 Raw FIT

Raw FIT is the FIT rate of the FPGA if the design uses every component. The device Reliability Report and the Quartus Prime Projected SEU FIT by Component Usage report both provide raw FIT data.

- The Reliability Report, available on the Altera web site, provides reliability data and testing procedures for Intel devices. The raw FIT data is not design specific.
- The Quartus Prime Projected SEU FIT by Component Usage report shows the raw FIT in the **Raw** column. The Quartus Prime software computes the FIT for each component using (component Mb × intrinsic FIT/Mb × Neutron Flux Multiplier) for the device family and process node. (For flipflops, “Mb” represents a million flipflops.)

For example, a CRAM FIT rate of 5,414 is the number of Mb of CRAM in a Stratix V GX A4 chip multiplied by the CRAM FIT/Mb. To give the worst-case raw FIT, the report assumes the maximum amount of CRAM that implements MLABs in the device. Thus, the CRAM raw FIT is the sum of CRAM and MLAB entries, which, in this example, is 5,414 + 2,767= 8,181. For M20K blocks, the raw FIT assumes that the design completely uses all 20 Kb of all M20K blocks.

Note: The Quartus Prime software counts device bits for target devices using different parameter information than the Reliability Report. Therefore, expect a ±5% variation in the Projected SEU FIT by Component Usage report **Raw** column compared to the Reliabilty Report data.

16.7.3 Utilized FIT

The **Utilized** column shows the FIT for the bits that the design actually uses. For example, if the design does not use a LUT or routing multiplexer, the **Utilized** column does not include those bits. Intel FPGAs can tolerate SEU events in unused resources; these events do not affect the FPGA. Therefore, you can safely ignore these bits for resiliency statistics.

The **Utilized** column discounts unused memory bits. For example, a 16 × 16 memory implemented in an M20K block uses only 256 bits of the total possible 20 Kb.

Comparing .smh Size to Utilized Bit Count

Although the `.smh` size correlates to the utilized bit counts, it is not exact. For example, the `.smh` indicates whether or not the design uses a particular ALM. However, it does not indicate how much of the ALM the design uses. Therefore, it is not an accurate indicator of FIT. For example, a design implementing one 4-LUT in a 6-LUT ALM shows only 16 out of 64 used CRAM bits in the Projected SEU FIT by Component report. In constrast, the `.smh` counts the full ALM. In this example, the `.smh` reports 60% of the LUTs and multiplexer blocks as used. The FIT report, which is much more detailed, correctly reports that the design uses only 30% of the FPGA's bits.

EDCRC Reported FIT

An FPGA's hard EDCRC recognizes that a bit flipped, and that this flip resulted in a non-zero CRC calculation. However, the FPGA cannot determine whether the design uses the bit. Thus, the reported error rate is based on raw CRAM FIT, not utilized FIT. To reduce this rate to the utilized and sensitive FIT, apply hierarchy tagging and sensitivity processing to your design. This process filters out unused CRAM by comparing the FPGA's error message register to the Quartus Prime generated `.smh`.

The Projected SEU FIT by Component report's **Utilized** CRAM FIT represents provable deflation of the FIT rate to account for CRAM upsets that do not matter to the design. Thus, the EDCRC incidence is always higher than the utilized FIT rate.

Note: The EDCRC flag and the Projected SEU FIT by Component report do not distinguish between bit upsets that are more important such as fundamental control logic or less important such as initialization logic that executes only once in the design. Apply hierarchy tags at the system level to filter out less important logic errors.

Small Designs

Raw FIT is always correct. In contrast, the utilized FIT only becomes accurate for designs that reasonably fill up the chosen device. FPGAs contain overhead, such as the configuration state machine, the clock network control logic, and the I/O calibration block. These infrastructure blocks contain flipflops, memories, and sometimes I/O configuration blocks. The Projected SEU FIT by Component report includes the constant overhead for GPIO and HSSI calibration circuitry for first I/O block or transceiver the design uses. Because of this overhead, the FIT of a 1-transceiver design is much higher than 1/10 the FIT of a 10-transceiver design. On the other hand, a trivial design such as "a single AND gate plus flipflop" could easily use so few bits that its CRAM FIT rate is 0.01, which the report rounds to zero.

Related Links

Altera Advanced SEU Detection IP Core User Guide

16.7.4 Mitigated FIT

You can lower FIT by reducing the observed FIT rate, such as by enabling ECC. You can also use the optional M20K ECC to mitigate FIT, as well as the (not optional) hard processor ECC and other hard IP such as memory controllers, PCIe, and I/O calibration blocks.

The Projected SEU FIT by Component Usage report's **w/ECC** column represents the FPGA's lowest guaranteed, provable FIT rate that the Quartus Prime software can calculate. ECC does not affect CRAM and flipflop rates; therefore, the data in the **w/ECC** column for these components is the same as the in **Utilized** column.

The ECC code strength varies with the device family. In Arria 10 devices, the M20K block can correct up to two errors, and the FIT rate beyond two (not corrected) is small enough to be negligible in the total.

An MLAB is simply a LAB configured with writable CRAM. However, when the Quartus Prime software configures the RAM as write enabled (MLAB), the MLAB has a slightly different FIT/Mb. The Projected SEU FIT by Component Usage report displays a FIT rate in the MLAB row when the design uses MLABs, otherwise the report accounts for the block's FIT in the CRAM row. During compilation, if the Quartus Prime software changes a LAB to an MLAB, the FIT accounting moves from the LAB row to the MLAB row.

The **w/ECC** column does not account for other forms of FIT protection in the design, such as designer-inserted parity, soft ECC blocks, bounds checking, system monitors, triple-module redundancy, or the impact of higher-level protocols on general fault tolerance. Additionally, it does not account for single event effects that occur in the logic but the design never reads or notices. For example, if you implement a non-ECC FIFO function 512 bits deep and an SEU event occurs outside of the front and back pointers, the application does not observe the SEU event. However, the report accounts for the full 512 bit deep memory and includes it in the **w/ECC** FIT rate. Designers often combine these factors into general deflation factors (called architectural vulnerability factors or AVF) based on knowledge of their design. Designers use AVF factors as low (aggressive) as 5% and as high (conservative) as 50% based on experience, fault-injection or neutron beam testing, or high-level system monitors.

16.7.5 Architectural Vulnerability Factor

10% SEFI factors are a typical specification to deflate the raw FIT to that observed in practice. For convenience, the last two columns in the Projected SEU FIT by Component Usage report show AVF deflations for SEFI of 50% (conservative) and 25% (reasonable).

SEFI represents a combination of factors. A utilization + ECC factor of 40% and AVF of .25% thus represents a global SEFI factor of 10% (because 0.4 × 0.25 = 0.1). An end-to-end SEFI factor of 10% is typical for a full design.

Related Links

Understanding Single Event Functional Interrupts in FPGA Designs White Paper

16.7.6 Enabling the Projected SEU FIT by Component Usage Report

The Quartus Prime Fitter generates the Projected SEU FIT by Component Usage report. The Quartus Prime software only generates reports for designs that successfully pass place and route.

To enable the report:

1. Obtain and install the SEU license.
2. Add the following assignments to your project's `.qsf`:
   - `set_global_assignment -name ENABLE_ADV_SEU_DETECTION ON`
   - `set_global_assignment -name SEU_FIT_REPORT ON`

16.8 Triple-Module Redundancy

Use Triple-Module Redundancy (TMR) if your system cannot suffer downtime due to SEU. TMR is an established SEU mitigation technique for improving hardware fault tolerance. A TMR design has three identical instances of hardware with voting hardware at the output. If an SEU affects one of the hardware instances, the voting logic notes the majority output. This operation masks malfunctioning hardware.

With TMR, your design does not suffer downtime in the case of a single SEU; if the system detects a faulty module, the system can scrub the error by reprogramming the module. The error detection and correction time is many orders of magnitude less than the MTBF of SEU events. Therefore, the system can repair a soft interrupt before another SEU affects another instance in the TMR application.

The disadvantage of TMR is its hardware resource cost: it requires three times as much hardware in addition to voting logic. You can minimize this hardware cost by implementing TMR for only the most critical parts of your design.

There are several automated ways to generate TMR designs by automatically replicating designated functions and synthesizing the required voting logic. Synopsys offers automated TMR synthesis.

16.9 Evaluating Your System's Response to Functional Upsets

Because SEU can randomly strike any memory element, perform system testing to ensure a comprehensive recovery response. The Quartus Prime software includes the Fault Injection Debugger to aid in SEU recovery.

The Fault Injection Debugger works together with the Altera Fault Injection IP core. You instantiate the Altera Fault Injection IP core in your FPGA design to use the debugging feature. During debugging, the IP core flips a CRAM bit by dynamically reconfiguring the frame containing the CRAM bit.

You use the Fault Injection Debugger's GUI or the command line to operate the FPGA in your system and inject random CRAM bit flips. The debugger helps test how the FPGA and the system detect and recover from SEU. Observe your FPGA and your system recover from these simulated SEU strikes. Then, refine your FPGA and system recovery sequence.

Related Links

Debugging Single Event Upsets Using the Fault Injection Debugger

16.10 Document Revision History

Table 257. Document Revision History

Date	Version	Changes
2016.10.31	16.1.0	• Implemented Intel rebranding. • Removed incorrect link to Reliability Report. • Added MAX 10 and Cylone IV to list of devices supporting Projected SEU FIT by Component Usage Report.
2016.05.24	16.0.1	• Corrected the steps to enable the SEU FIT reports.
2016.05.03	16.0.0	• Documented the new SEU FIT reports. • Inconsequential wording changes for conformance to style.
2015.11.02	15.1.0	• Changed instances of *Quartus II* to *Quartus Prime*.
June 2014	2014.06.30	• Updated formatting. • Added "Mitigating SEU Effects in Embedded User RAM" section. • Added "Altera Advanced SEU Detection IP Core" section.
November 2012	2012.11.01	• Preliminary release.

17 Optimizing the Design Netlist

This chapter describes how you can use the Quartus Prime Netlist Viewers to analyze and debug your designs.

As FPGA designs grow in size and complexity, the ability to analyze, debug, optimize, and constrain your design is critical. With today's advanced designs, several design engineers are involved in coding and synthesizing different design blocks, making it difficult to analyze and debug the design. The Quartus Prime RTL Viewer and Technology Map Viewer provide powerful ways to view your initial and fully mapped synthesis results during the debugging, optimization, and constraint entry processes.

Related Links

- When to Use the Netlist Viewers: Analyzing Design Problems on page 981
- Introduction to the User Interface on page 984
  The Netlist Viewer is a graphical user-interface for viewing and manipulating nodes and nets in the netlist.
- Quartus Prime Design Flow with the Netlist Viewers on page 982
  When you first open one of the Netlist Viewers after compiling the design, a preprocessor stage runs automatically before the Netlist Viewer opens.
- RTL Viewer Overview on page 983
  The RTL Viewer allows you to view a register transfer level (RTL) graphical representation of your Quartus Prime Pro Edition synthesis results or your third-party netlist file in the Quartus Prime software.
- Technology Map Viewer Overview on page 984
  The Quartus Prime Technology Map Viewer provides a technology-specific, graphical representation of your design after Analysis and Synthesis or after the Fitter has mapped your design into the target device.
- Filtering in the Schematic View on page 994
  Filtering allows you to filter out nodes and nets in your netlist to view only the logic elements of interest to you.
- Cross-Probing to a Source Design File and Other Quartus Prime Windows on page 999
  The RTL Viewer and Technology Map Viewer allow you to cross-probe to the source design file and to various other windows in the Quartus Prime software.
- Cross-Probing to the Netlist Viewers from Other Quartus Prime Windows on page 1000
  You can cross-probe to the RTL Viewer and Technology Map Viewer from other windows in the Quartus Prime software. You can select one or more nodes or nets in another window and locate them in one of the Netlist Viewers.
- Viewing a Timing Path on page 1001
  You can cross-probe from a report panel in the TimeQuest Timing Analyzer to see a visual representation of a timing path.

17.1 When to Use the Netlist Viewers: Analyzing Design Problems

You can use the Netlist Viewers to analyze and debug your design. The following simple examples show how to use the RTL Viewer and Technology Map Viewer to analyze problems encountered in the design process.

Using the RTL Viewer is a good way to view your initial synthesis results to determine whether you have created the necessary logic, and that the logic and connections have been interpreted correctly by the software. You can use the RTL Viewer to check your design visually before simulation or other verification processes. Catching design errors at this early stage of the design process can save you valuable time.

If you see unexpected behavior during verification, use the RTL Viewer to trace through the netlist and ensure that the connections and logic in your design are as expected. Viewing your design helps you find and analyze the source of design problems. If your design looks correct in the RTL Viewer, you know to focus your analysis on later stages of the design process and investigate potential timing violations or issues in the verification flow itself.

You can use the Technology Map Viewer to look at the results at the end of Analysis and Synthesis. If you have compiled your design through the Fitter stage, you can view your post-mapping netlist in the Technology Map Viewer (Post-Mapping) and your post-fitting netlist in the Technology Map Viewer. If you perform only Analysis and Synthesis, both the Netlist Viewers display the same post-mapping netlist.

In addition, you can use the RTL Viewer or Technology Map Viewer to locate the source of a particular signal, which can help you debug your design. Use the navigation techniques described in this chapter to search easily through your design. You can trace back from a point of interest to find the source of the signal and ensure the connections are as expected.

The Technology Map Viewer can help you locate post-synthesis nodes in your netlist and make assignments when optimizing your design. This functionality is useful when making a multicycle clock timing assignment between two registers in your design. Start at an I/O port and trace forward or backward through the design and through levels of hierarchy to find nodes of interest, or locate a specific register by visually inspecting the schematic.

Throughout your FPGA design, debug, and optimization stages, you can use all of the netlist viewers in many ways to increase your productivity while analyzing a design.

Related Links

- Quartus Prime Design Flow with the Netlist Viewers on page 982
  When you first open one of the Netlist Viewers after compiling the design, a preprocessor stage runs automatically before the Netlist Viewer opens.
- RTL Viewer Overview on page 983
  The RTL Viewer allows you to view a register transfer level (RTL) graphical representation of your Quartus Prime Pro Edition synthesis results or your third-party netlist file in the Quartus Prime software.
- Technology Map Viewer Overview on page 984
  The Quartus Prime Technology Map Viewer provides a technology-specific, graphical representation of your design after Analysis and Synthesis or after the Fitter has mapped your design into the target device.

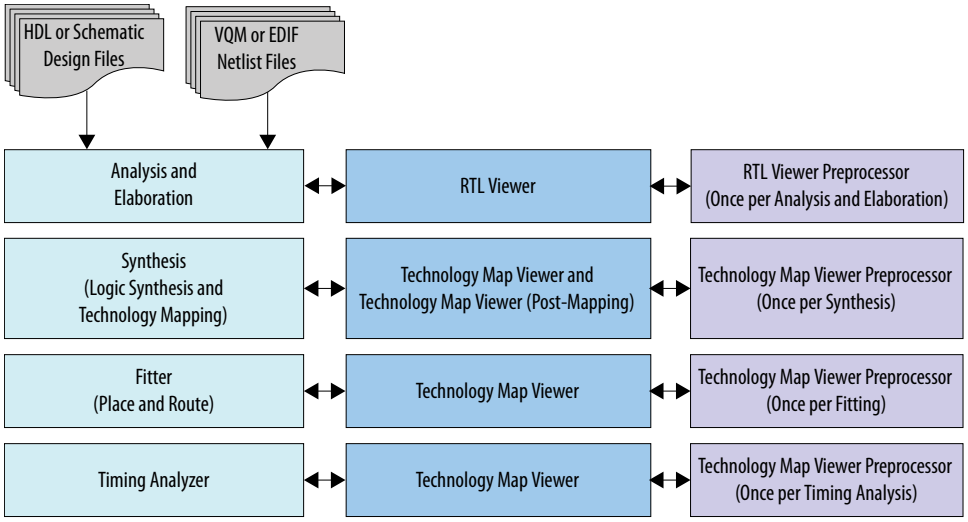
17.2 Quartus Prime Design Flow with the Netlist Viewers

When you first open one of the Netlist Viewers after compiling the design, a preprocessor stage runs automatically before the Netlist Viewer opens.

Click the link in the preprocessor process box to go to the **Settings ➤ Compilation Process Settings** page where you can turn on the **Run Netlist Viewers preprocessing during compilation** option. If you turn this option on, the preprocessing becomes part of the full project compilation flow and the Netlist Viewer opens immediately without displaying the preprocessing dialog.

Figure 292. Quartus Prime Design Flow Including the RTL Viewer and Technology Map Viewer

This figure shows how Netlist Viewers fit into the basic Quartus Prime design flow.



Before the Netlist Viewer can run the preprocessor stage, you must compile your design:

- To open the RTL Viewer first perform Analysis and Elaboration.
- To open the Technology Map Viewer (Post-Fitting) or the Technology Map Viewer (Post-Mapping), first perform Analysis and Synthesis.

The Netlist Viewers display the results of the last successful compilation.

- Therefore, if you make a design change that causes an error during Analysis and Elaboration, you cannot view the netlist for the new design files, but you can still see the results from the last successfully compiled version of the design files.
- If you receive an error during compilation and you have not yet successfully run the appropriate compilation stage for your project, the Netlist Viewer cannot be displayed; in this case, the Quartus Prime software issues an error message when you try to open the Netlist Viewer.

Note: If the Netlist Viewer is open when you start a new compilation, the Netlist Viewer closes automatically. You must open the Netlist Viewer again to view the new design netlist after compilation completes successfully.

17.3 RTL Viewer Overview

The RTL Viewer allows you to view a register transfer level (RTL) graphical representation of your Quartus Prime Pro Edition synthesis results or your third-party netlist file in the Quartus Prime software.

You can view results after Analysis and Elaboration when your design uses any supported Quartus Prime design entry method, including Verilog HDL Design Files (`.v`), SystemVerilog Design Files (`.sv`), VHDL Design Files (`.vhd`), AHDL Text Design Files (`.tdf`), or schematic Block Design Files (`.bdf`). You can also view the hierarchy of atom primitives (such as device logic cells and I/O ports) when your design uses a synthesis tool to generate a Verilog Quartus Mapping File (`.vqm`) or Electronic Design Interchange Format (`.edf`) file.

The RTL Viewer displays a schematic view of the design netlist after Analysis and Elaboration or netlist extraction is performed by the Quartus Prime software, but before technology mapping and any synthesis or fitter optimizations. This view a preliminary pre-optimization design structure and closely represents your original source design.

- If you synthesized your design with the Quartus Prime Pro Edition synthesis, this view shows how the Quartus Prime software interpreted your design files.
- If you use a third-party synthesis tool, this view shows the netlist written by your synthesis tool.

While displaying your design, the RTL Viewer optimizes the netlist to maximize readability:

- Removes logic with no fan-out (unconnected output) or fan-in (unconnected inputs) from the display.
- Hides default connections such as V_{CC} and GND.
- Groups pins, nets, wires, module ports, and certain logic into buses where appropriate.
- Groups constant bus connections are grouped.
- Displays values in hexadecimal format.
- Converts NOT gates into bubble inversion symbols in the schematic.
- Merges chains of equivalent combinational gatesinto a single gate; for example, a 2-input AND gate feeding a 2-input AND gate is converted to a single 3-input AND gate.

To run the RTL Viewer for a Quartus Prime project, first analyze the design to generate an RTL netlist. To analyze the design and generate an RTL netlist, click **Processing ➤ StartStart Analysis & Elaboration**. You can also perform a full compilation on any process that includes the initial Analysis and Elaboration stage of the Quartus Prime compilation flow.

To open the RTL Viewer, click **Tools ➤ Netlist ViewersRTL Viewer**.

Related Links

Introduction to the User Interface on page 984
The Netlist Viewer is a graphical user-interface for viewing and manipulating nodes and nets in the netlist.

17.4 Technology Map Viewer Overview

The Quartus Prime Technology Map Viewer provides a technology-specific, graphical representation of your design after Analysis and Synthesis or after the Fitter has mapped your design into the target device.

The Technology Map Viewer shows the hierarchy of atom primitives (such as device logic cells and I/O ports) in your design. For supported device families, you can also view internal registers and look-up tables (LUTs) inside logic cells (LCELLs), and registers in I/O atom primitives.

Where possible, the Quartus Prime software maintains the port names of each hierarchy throughout synthesis. However, the software may change or remove port names from the design. For example, if a port is unconnected or driven by GND or V_{CC}, the software removes it during synthesis. If a port name changes, the software assigns a related user logic name in the design or a generic port name such as `IN1` or `OUT1`.

You can view your Quartus Prime technology-mapped results after synthesis, fitting, or timing analysis. To run the Technology Map Viewer for a Quartus Prime project, on the Processing menu, point to **Start** and click **Start Analysis & Synthesis** to synthesize and map the design to the target technology. At this stage, the Technology Map Viewer shows the same post-mapping netlist as the Technology Map Viewer (Post-Mapping). You can also perform a full compilation, or any process that includes the synthesis stage in the compilation flow.

If you have completed the Fitter stage, the Technology Map Viewer shows the changes made to your netlist by the Fitter, such as physical synthesis optimizations, while the Technology Map Viewer (Post-Mapping) shows the post-mapping netlist. If you have completed the Timing Analysis stage, you can locate timing paths from the Timing Analyzer report in the Technology Map Viewer.

To open the Technology Map Viewer, on the Tools menu, point to **Netlist Viewers** and click **Technology Map Viewer (Post-Fitting)** or **Technology Map Viewer (Post Mapping)**.

Related Links

- View Contents of Nodes in the Schematic View on page 995
  In the RTL Viewer and the Technology Map Viewer, you can view the contents of nodes to see their underlying implementation details.
- Viewing a Timing Path on page 1001
  You can cross-probe from a report panel in the TimeQuest Timing Analyzer to see a visual representation of a timing path.
- Introduction to the User Interface on page 984
  The Netlist Viewer is a graphical user-interface for viewing and manipulating nodes and nets in the netlist.

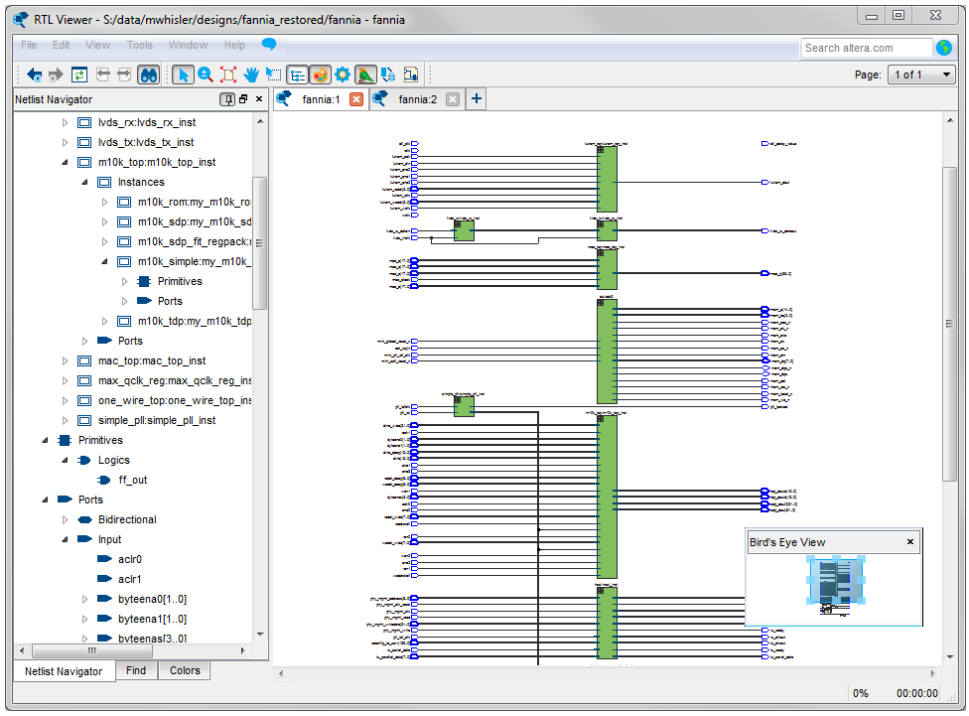
17.5 Introduction to the User Interface

The Netlist Viewer is a graphical user-interface for viewing and manipulating nodes and nets in the netlist.

The RTL Viewer and Technology Map Viewer each consist of these main parts:

- The **Netlist Navigator** pane—displays a representation of the project hierarchy.
- The **Find** pane—allows you to find and locate specific design elements in the schematic view.
- The **Properties** pane displays the properties of the selected block when you select **Properties** from the shortcut menu.
- The schematic view—displays a graphical representation of the internal structure of your design.

Figure 293. RTL Viewer

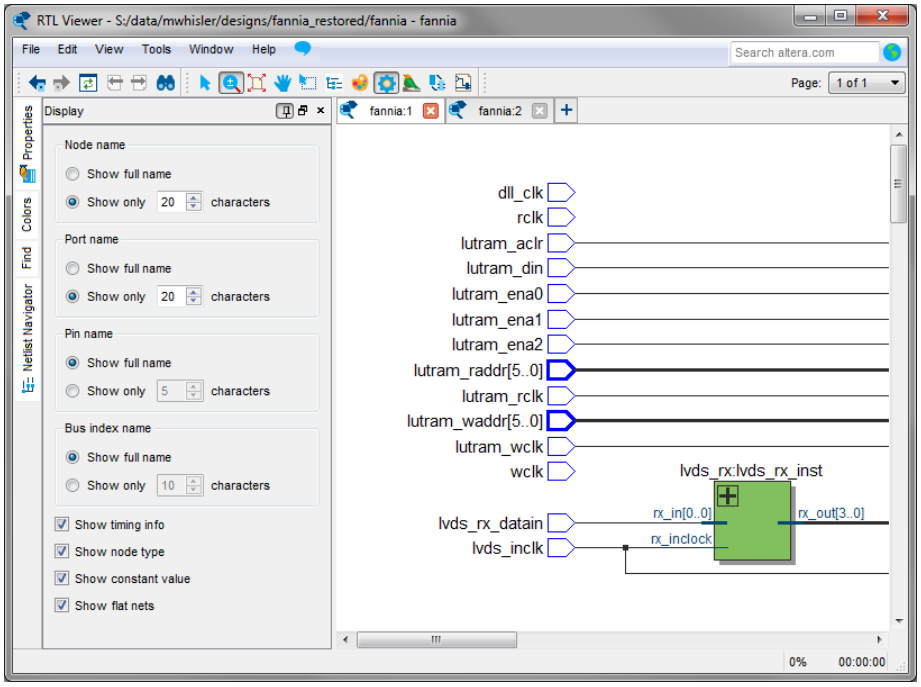


Netlist Viewers also contain a toolbar that provides tools to use in the schematic view.

- Use the **Back** and **Forward** buttons to switch between schematic views. You can go forward only if you have not made any changes to the view since going back. These commands do not undo an action, such as selecting a node. The Netlist Viewer caches up to ten actions including filtering, hierarchy navigation, netlist navigation, and zoom actions.
- The **Refresh** button to restore the schematic view and optimizes the layout. **Refresh** does not reload the database if you change your design and recompile.
- Click the **Find** button opens and closes the **Find** pane.
- Click the **Selection Tool** and **Zoom Tool** buttons to toggle between the selection mode and zoom mode.
- Click the **Fit in Page** button resets the schematic view to encompass the entire design.
- Use the **Hand Tool** to change the focus of the veiwer without changing the perspective.

- Click the **Area Selection Tool** to drag a selection box around ports, pins, and nodes in an area.
- Click the **Netlist Navigator** button to open or close the **Netlist Navigator** pane.
- Click the **Color Settings** button to open the **Colors** pane where you can customize the Netlist Viewer color scheme.
- Click the **Display Settings** button to open the **Display** pane where you can specify the following settings:
  - **Show full name** or **Show only <*n*> characters**. You can specify this separately for **Node name**, **Port name**, **Pin name**, or **Bus name**.
  - Turn **Show timing info** on or off.
  - Turn **Show node type** on or off.
  - Turn **Show constant value** on or off.
  - Turn **Show flat nets** on or off.

Figure 294. Display Settings



- The **Bird's Eye View** button opens the **Bird's Eye View** window which displays a miniature version of your design and allows you to navigate within the design and adjust the magnification in the schematic view quickly.
- The **Show/Hide Instance Pins** button can toggle the display of instance pins not displayed by functions such as cross-probing between a Netlist Viewer and TimeQuest. You can also use it to hide unconnected instance pins when filtering a node results in large numbers of unconnected or unused pins. Instance pins are hidden by default.
- The **Show Netlist on One Page** button displays the netlist on a single page if the Netlist Veiwer has split the design across several pages. This can make netlist tracing easier.

You can have only one RTL Viewer, one Technology Map Viewer (Post-Fitting), ande one Technology Map Viewer (Post-Mapping) window open at the same time, although each window can show multiple pages, each with multiple tabs. For example, you cannot have two RTL Viewer windows open at the same time.

Related Links

- RTL Viewer Overview on page 983
  The RTL Viewer allows you to view a register transfer level (RTL) graphical representation of your Quartus Prime Pro Edition synthesis results or your third-party netlist file in the Quartus Prime software.
- Technology Map Viewer Overview on page 984
  The Quartus Prime Technology Map Viewer provides a technology-specific, graphical representation of your design after Analysis and Synthesis or after the Fitter has mapped your design into the target device.
- Netlist Navigator Pane on page 987
  The **Netlist Navigator** pane displays the entire netlist in a tree format based on the hierarchical levels of the design. In each level, similar elements are grouped into subcategories.
- Netlist Viewers Find Pane on page 989
- Properties Pane on page 988
  You can view the properties of an instance or primitive using the **Properties** pane.

17.5.1 Netlist Navigator Pane

The **Netlist Navigator** pane displays the entire netlist in a tree format based on the hierarchical levels of the design. In each level, similar elements are grouped into subcategories.

You can use the **Netlist Navigator** pane to traverse through the design hierarchy to view the logic schematic for each level. You can also select an element in the **Netlist Navigator** to highlight in the schematic view.

Note: Nodes inside atom primitives are not listed in the **Netlist Navigator** pane.

For each module in the design hierarchy, the **Netlist Navigator** pane displays the applicable elements listed in the following table. Click the "**+**" icon to expand an element.

Table 258. Netlist Navigator Pane Elements

Elements	Description
Instances	Modules or instances in the design that can be expanded to lower hierarchy levels.
Primitives	Low-level nodes that cannot be expanded to any lower hierarchy level. These primitives include: • Registers and gates that you can view in the RTL Viewer when using Quartus Prime Pro Edition synthesis. • Logic cell atoms in the Technology Map Viewer or in the RTL Viewer when using a VQM or EDIF from third-party synthesis software

continued...

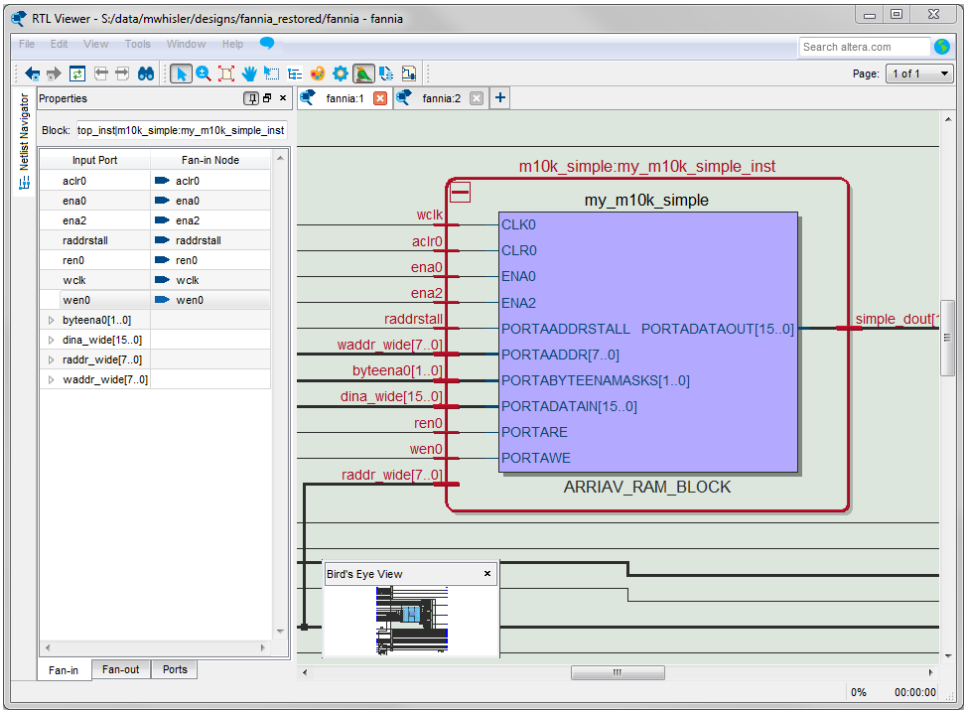
Elements	Description
	In the Technology Map Viewer, you can view the internal implementation of certain atom primitives, but you cannot traverse into a lower-level of hierarchy.
Ports	The I/O ports in the current level of hierarchy. • Pins are device I/O pins when viewing the top hierarchy level and are I/O ports of the design when viewing the lower-levels. • When a pin represents a bus or an array of pins, expand the pin entry in the list view to see individual pin names.

17.5.2 Properties Pane

You can view the properties of an instance or primitive using the **Properties** pane.

Figure 295. Properties Pane

To view the properties of an instance or primitive in the RTL Viewer or Technology Map Viewer, right-click the node and click **Properties**.



The **Properties** pane contains tabs with the following information about the selected node:

- The **Fan-in** tab displays the **Input port** and **Fan-in Node**.
- The **Fan-out** tab displays the **Output port** and **Fan-out Node**.
- The **Parameters** tab displays the **Parameter Name** and **Values** of an instance.
- The **Ports** tab displays the **Port Name** and **Constant** value (for example, V_{CC} or GND). The possible value of a port are listed below.

Table 259. Possible Port Values

Value	Description
V_{CC}	The port is not connected and has V_{CC} value (tied to V_{CC})
GND	The port is not connected and has GND value (tied to GND)
--	The port is connected and has value (other than V_{CC} or GND)
Unconnected	The port is not connected and has no value (hanging)

If the selected node is an atom primitive, the **Properties** pane displays a schematic of the internal logic.

17.5.3 Netlist Viewers Find Pane

You can narrow the range of the search process by setting the following options in the **Find** pane:

- Click **Browse** in the **Find** pane to specify the hierarchy level of the search. In the **Select Hierarchy Level** dialog box, select the particular instance you want to search.
- Turn on the **Include subentities** option to include child hierarchies of the parent instance during the search.
- Click **Options** to open the **Find Options** dialog box. Turn on **Instances**, **Nodes**, **Ports**, or any combination of the three to further refine the parameters of the search.

When you click the **List** button, a progress bar appears below the **Find** box.

All results that match the criteria you set are listed in a table. When you double-click an item in the table, the related node is highlighted in red in the schematic view.

17.6 Schematic View

The schematic view is shown on the right side of the RTL Viewer and Technology Map Viewer. The schematic view contains a schematic representing the design logic in the netlist. This view is the main screen for viewing your gate-level netlist in the RTL Viewer and your technology-mapped netlist in the Technology Map Viewer.

The RTL Viewer and Technology Map Viewer attempt to display schematic in a single page view by default. If the schematic crosses over to several pages, you can highlight a net and use connectors to trace the signal in a single page.

17.6.1 Display Schematics in Multiple Tabbed View

The RTL Viewer and Technology Map Viewer support multiple tabbed views.

With multiple tabbed view, schematics can be displayed in different tabs. Selection is independent between tabbed views, but selection in the tab in focus is synchronous with the Netlist Navigator pane.

To create a new blank tab, click the **New Tab** button at the end of the tab row . You can now drag a node from the **Netlist Navigator** pane into the schematic view.

Right-click in a tab to see a shortcut menu to perform the following actions:

- Create a blank view with **New Tab**
- Create a **Duplicate Tab** of the tab in focus
- Choose to **Cascade Tabs**
- Choose to **Tile Tabs**
- Choose **Close Tab** to close the tab in focus
- Choose **Close Other Tabs** to close all tabs except the tab in focus

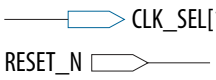
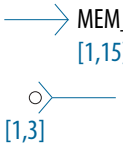
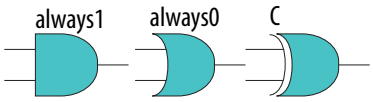
17.6.2 Schematic Symbols

The symbols for nodes in the schematic represent elements of your design netlist. These elements include input and output ports, registers, logic gates, Intel primitives, high-level operators, and hierarchical instances.

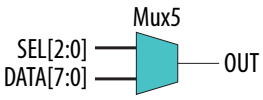
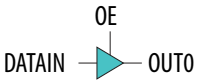
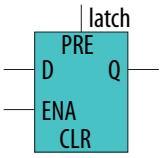
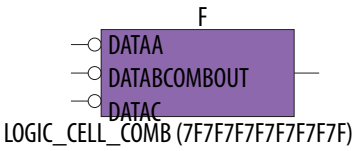
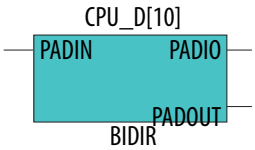
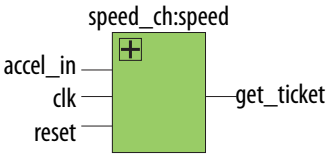
Note: The logic gates and operator primitives appear only in the RTL Viewer. Logic in the Technology Map Viewer is represented by atom primitives, such as registers and LCELLs.

Table 260. Symbols in the Schematic View

This table lists and describes the primitives and basic symbols that you can display in the schematic view of the RTL Viewer and Technology Map Viewer.

Symbol	Description
I/O Ports 	An input, output, or bidirectional port in the current level of hierarchy. A device input, output, or bidirectional pin when viewing the top-level hierarchy. The symbol can also represent a bus. Only one wire is shown connected to the bidirectional symbol, representing the input and output paths. Input symbols appear on the left-most side of the schematic. Output and bidirectional symbols appear on the right-most side of the schematic.
I/O Connectors 	An input or output connector, representing a net that comes from another page of the same hierarchy. To go to the page that contains the source or the destination, double-click on the connector to jump to the appropriate page.
OR, AND, XOR Gates 	An OR, AND, or XOR gate primitive (the number of ports can vary). A small circle (bubble symbol) on an input or output port indicates the port is inverted.
MULTIPLEXER	A multiplexer primitive with a selector port that selects between port `0` and port `1`. A multiplexer with more than two inputs is displayed as an operator.

continued...

Symbol	Description
	
BUFFER 	A buffer primitive. The figure shows the tri-state buffer, with an inverted output enable port. Other buffers without an enable port include LCELL, SOFT, CARRY, and GLOBAL. The NOT gate and EXP expander buffers use this symbol without an enable port and with an inverted output port.
LATCH 	A latch/DFF (data flipflop) primitive. A DFF has the same ports as a latch and a clock trigger. The other flipflop primitives are similar: • DFFEA (data flipflop with enable and asynchronous load) primitive with additional ALOAD asynchronous load and ADATA data signals • DFFEAS (data flipflop with enable and synchronous and asynchronous load), which has ASDATA as the secondary data port
Atom Primitive 	An atom primitive. The symbol displays the atom name, the port names, and the atom type. The blue shading indicates an atom primitive for which you can view the internal details.
Other Primitive 	Any primitive that does not fall into the previous categories. Primitives are low-level nodes that cannot be expanded to any lower hierarchy. The symbol displays the port names, the primitive or operator type, and its name.
Instance 	An instance in the design that does not correspond to a primitive or operator (a user-defined hierarchy block). The symbol displays the port name and the instance name.
Ecrypted Instance	A user-defined encrypted instance in the design. The symbol displays the instance name. You cannot open the schematic for the lower-level hierarchy, because the source design is encrypted.

continued...

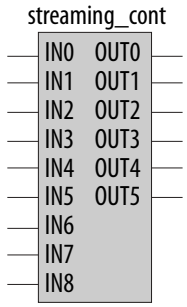
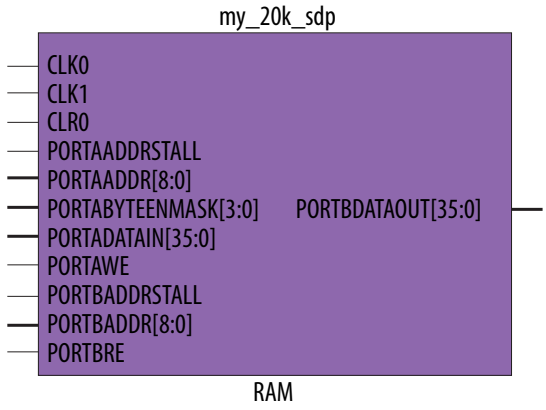
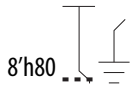
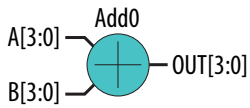
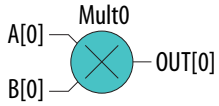
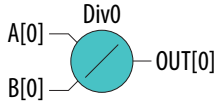
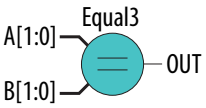
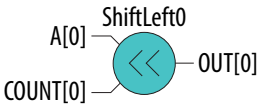
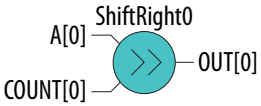
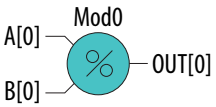
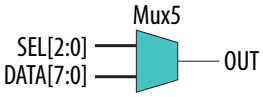
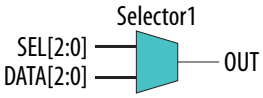
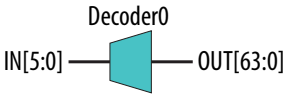
Symbol	Description
	
RAM 	A synchronous memory instance with registered inputs and optionally registered outputs. The symbol shows the device family and the type of memory block. This figure shows a true dual-port memory block in a Stratix M-RAM block.
Constant 	A constant signal value that is highlighted in gray and displayed in hexadecimal format by default throughout the schematic.

Table 261. Operator Symbols in the RTL Viewer Schematic View

The following lists and describes the additional higher level operator symbols in the RTL Viewer schematic view.

Symbol	Description
	An adder operator: OUT = A + B
	A multiplier operator: OUT = A ¥ B
	A divider operator: OUT = A / B

continued...

Symbol	Description
	Equals
	A left shift operator: `OUT = (A << COUNT)`
	A right shift operator: `OUT = (A >> COUNT)`
	A modulo operator: `OUT = (A%B)`
	A less than comparator: `OUT = (A<:B:A>B)`
	A multiplexer: `OUT = DATA [SEL]` The data range size is $2^{\text{sel range size}}$
	A selector: A multiplexer with one-hot select input and more than two input signals
	A binary number decoder: `OUT = (binary_number (IN) == x)` for x = 0 to x = $2^{(n+1)} - 1$

Related Links

- Partition the Schematic into Pages on page 999
  For large design hierarchies, the RTL Viewer and Technology Map Viewer partition your netlist into multiple pages in the schematic view.
- Follow Nets Across Schematic Pages on page 999
  Input and output connector symbols indicate nodes that connect across pages of the same hierarchy. Double-click a connector to trace the net to the next page of the hierarchy.

17.6.3 Select Items in the Schematic View

To select an item in the schematic view, ensure that the **Selection Tool** is enabled in the Netlist Viewer toolbar (this tool is enabled by default). Click an item in the schematic view to highlight it in red.

Select multiple items by pressing the Shift key while selecting with your mouse.

Items selected in the schematic view are automatically selected in the **Netlist Navigator** pane. The folder then expands automatically if it is required to show the selected entry; however, the folder does not collapse automatically when you are not using or you have deselected the entries.

When you select a hierarchy box, node, or port in the schematic view, the item is highlighted in red but none of the connecting nets are highlighted. When you select a net (wire or bus) in the schematic view, all connected nets are highlighted in red.

Once you have selected an item, you can perform different actions on it based on the contents of the shortcut menu which appears when you right-click on your selection.

Related Links

Netlist Navigator Pane on page 987
The **Netlist Navigator** pane displays the entire netlist in a tree format based on the hierarchical levels of the design. In each level, similar elements are grouped into subcategories.

17.6.4 Shortcut Menu Commands in the Schematic View

When you right-click on an instance or primitive selected in the schematic view, the Netlist Viewer displays a shortcut menu.

If the selected item is a node, you see the following options:

- Click **Expand to Upper Hierarchy** to displays the parent hierarchy of the node in focus.
- Click **Copy ToolTip** to copy the selected item name to the clipboard. This command does not work on nets.
- Click **Hide Selection** to remove the selected item from the schematic view. This command does not delete the item from the design, merely masks it in the current view.
- Click **Filtering** to display a sub-menu with options for filtering your selection.

17.6.5 Filtering in the Schematic View

Filtering allows you to filter out nodes and nets in your netlist to view only the logic elements of interest to you.

You can filter your netlist by selecting hierarchy boxes, nodes, or ports of a node, that are part of the path you want to see. The following filter commands are available:

- **Sources**—Displays the sources of the selection.
- **Destinations**—Displays the destinations of the selection.
- **Sources & Destinations**—displays the sources and destinations of the selection.
- **Selected Nodes**—Displays only the selected nodes.

- **Between Selected Nodes**—Displays nodes and connections in the path between the selected nodes .
- **Bus Index**—Displays the sources or destinations for one or more indices of an output or input bus port .
- **Filtering Options**—Displays the **Filtering Options** dialog box:
  - **Stop filtering at register**—Turning this option on directs the Netlist Viewer to filter out to the nearest register boundary.
  - **Filter across hierarchies**—Turning this option on directs the Netlist Viewer to filter across hierarchies.
  - **Maximum number of hierarchy levels**—Sets the maximum number of hierarchy levels displayed in the schematic view.

To filter your netlist, select a hierarchy box, node, port, net, or state node, right-click in the window, point to **Filter** and click the appropriate filter command. The Netlist Viewer generates a new page showing the netlist that remains after filtering.

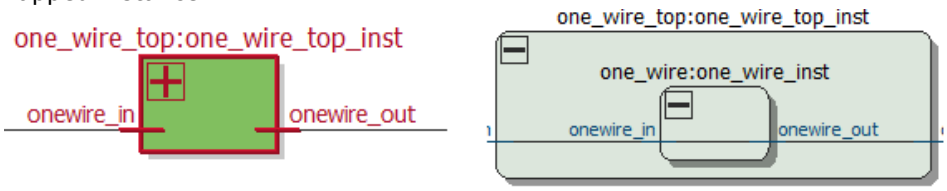
17.6.6 View Contents of Nodes in the Schematic View

In the RTL Viewer and the Technology Map Viewer, you can view the contents of nodes to see their underlying implementation details.

You can view LUTs, registers, and logic gates. You can also view the implementation of RAM and DSP blocks in certain devices in the RTL Viewer or Technology Map Viewer. In the Technology Map Viewer, you can view the contents of primitives to see their underlying implementation details.

Figure 296. Wrapping and Unwrapping Objects

If you can unwrap the contents of an instance, a plus symbol appears in the upper right corner of the object in the schematic view. To wrap the contents (and revert to the compact format), click the minus symbol in the upper right corner of the unwrapped instance.



Note: In the schematic view, the internal details in an atom instance cannot be selected as individual nodes. Any mouse action on any of the internal details is treated as a mouse action on the atom instance.

Figure 297. Nodes with Connections Outside the Hierarchy

In some cases, the selected instance connects to something outside the visible level of the hierarchy in the schematic view. In this case, the net appears as a dotted line. Double-click on the dotted line to expand the view to display the destination of the connection .

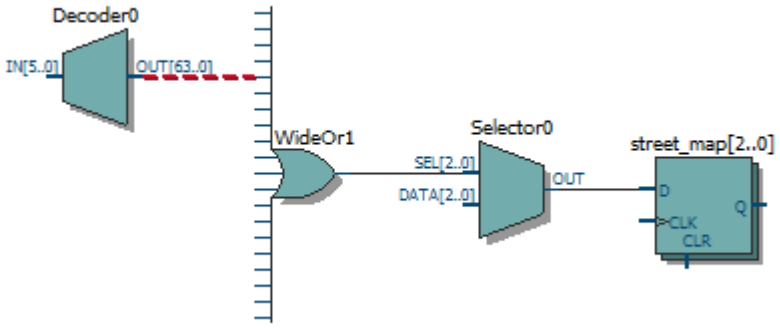


Figure 298. Display Nets Across Hierarchies

In cases where the net connects to an instance outside the hierarchy, you can select the net, and unwrap the node to see the destination ports.

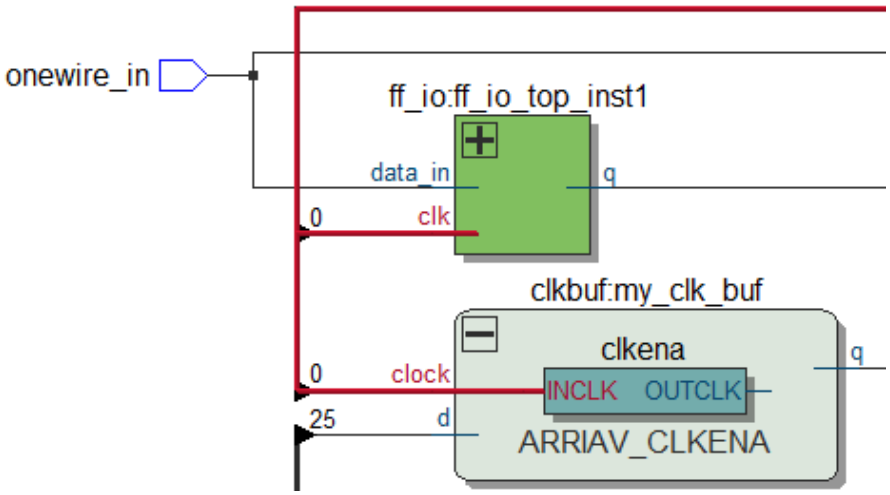
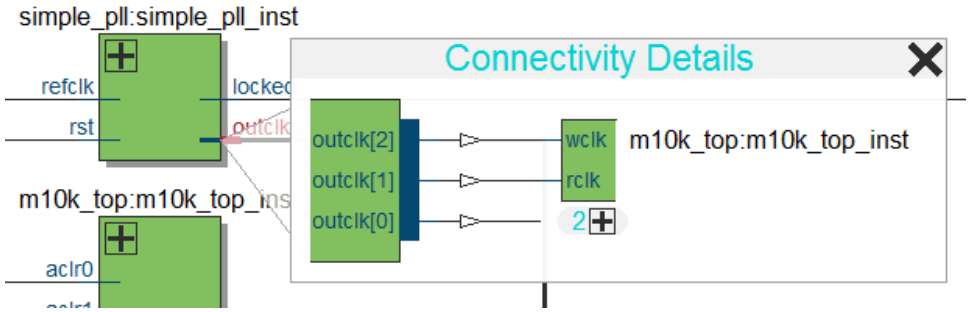


Figure 299. Show Connectivity Details

You can select a bus port or bus pin and click **Connectivity Details** in the context menu for that object.



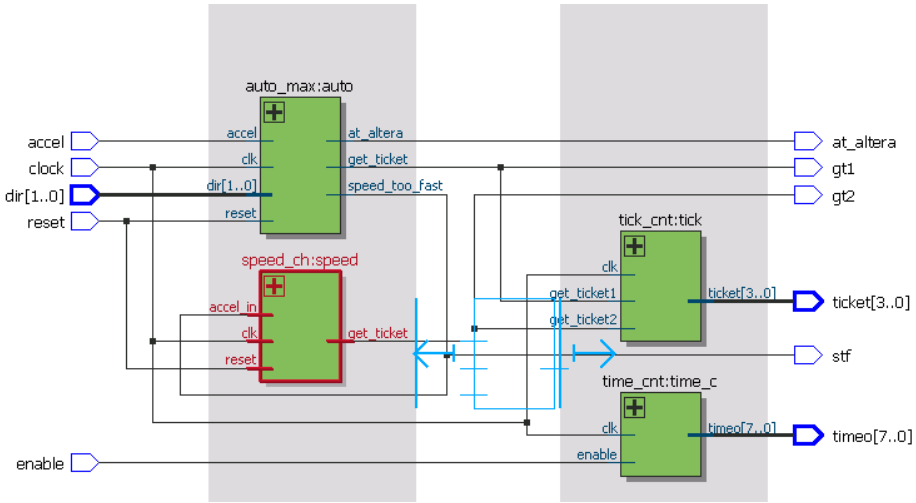
You can double-click on objects in the **Connectivity Details** window to navigate to them quickly. If the plus symbol appears, you can further unwrap objects in the view. This can be very useful when tracing a signal in a complex netlist.

17.6.7 Moving Nodes in the Schematic View

You can drag and drop items in the schematic view to rearrange them.

Figure 300. Drag and Drop Movement of Nodes

To move a node from one area of the netlist to another, select the node and hold down the Shift key. Legal placements appear as shaded areas within the hierarchy. Click to drop the selected node.



Right-click and click on **Refresh** to restore the schematic view to its default arrangement.

17.6.8 View LUT Representations in the Technology Map Viewer

You can view different representations of a LUT by right-clicking the selected LUT and clicking **Properties**.

You can view the LUT representations in the following three tabs in the **Properties** dialog box:

- The **Schematic** tab—the equivalent gate representations of the LUT.
- The **Truth Table** tab—the truth table representations.

Related Links

Properties Pane on page 988
You can view the properties of an instance or primitive using the **Properties** pane.

17.6.9 Zoom Controls

Use the Zoom Tool in the toolbar, or mouse gestures, to control the magnification of your schematic on the View menu.

By default, the Netlist Viewer displays most pages sized to fit in the window. If the schematic page is very large, the schematic is displayed at the minimum zoom level, and the view is centered on the first node. Click **Zoom In** to view the image at a larger size, and click **Zoom Out** to view the image (when the entire image is not displayed) at a smaller size. The **Zoom** command allows you to specify a magnification percentage (100% is considered the normal size for the schematic symbols).

You can use the Zoom Tool on the Netlist Viewer toolbar to control magnification in the schematic view. When you select the Zoom Tool in the toolbar, clicking in the schematic zooms in and centers the view on the location you clicked. Right-click in the schematic to zoom out and center the view on the location you clicked. When you select the Zoom Tool, you can also zoom into a certain portion of the schematic by selecting a rectangular box area with your mouse cursor. The schematic is enlarged to show the selected area.

Within the schematic view, you can also use the following mouse gestures to zoom in on a specific section:

- **zoom in**—Dragging a box around an area starting in the upper-left and dragging to the lower right zooms in on that area.
- **zoom -0.5**—Dragging a line from lower-left to upper-right zooms out 0.5 levels of magnification.
- **zoom 0.5**—Dragging a line from lower-right to upper-left zooms in 0.5 levels of magnification.
- **zoom fit**—Dragging a line from upper-right to lower-left fits the schematic view in the page.

Related Links

Filtering in the Schematic View on page 994
Filtering allows you to filter out nodes and nets in your netlist to view only the logic elements of interest to you.

17.6.10 Navigating with the Bird's Eye View

To open the Bird's Eye View, on the View menu, click **Bird's Eye View**, or click on the **Bird's Eye View** icon in the toolbar.

Viewing the entire schematic can be useful when debugging and tracing through a large netlist. The Quartus Prime software allows you to quickly navigate to a specific section of the schematic using the Bird's Eye View feature, which is available in the RTL Viewer and Technology Map Viewer.

The Bird's Eye View shows the current area of interest:

- Select an area by clicking and dragging the indicator or right-clicking to form a rectangular box around an area.
- Click and drag the rectangular box to move around the schematic.
- Resize the rectangular box to zoom-in or zoom-out in the schematic view.

17.6.11 Partition the Schematic into Pages

For large design hierarchies, the RTL Viewer and Technology Map Viewer partition your netlist into multiple pages in the schematic view.

When a hierarchy level is partitioned into multiple pages, the title bar for the schematic window indicates which page is displayed and how many total pages exist for this level of hierarchy. The schematic view displays this as **Page** <*current page number*> **of** <*total number of pages*>.

Related Links

Introduction to the User Interface on page 984
The Netlist Viewer is a graphical user-interface for viewing and manipulating nodes and nets in the netlist.

17.6.12 Follow Nets Across Schematic Pages

Input and output connector symbols indicate nodes that connect across pages of the same hierarchy. Double-click a connector to trace the net to the next page of the hierarchy.

Note: After you double-click to follow a connector port, the Netlist Viewer opens a new page, which centers the view on the particular source or destination net using the same zoom factor as the previous page. To trace a specific net to the new page of the hierarchy, Intel recommends that you first select the necessary net, which highlights it in red, before you double-click to navigate across pages.

Related Links

Schematic Symbols on page 990
The symbols for nodes in the schematic represent elements of your design netlist. These elements include input and output ports, registers, logic gates, Intel primitives, high-level operators, and hierarchical instances.

17.7 Cross-Probing to a Source Design File and Other Quartus Prime Windows

The RTL Viewer and Technology Map Viewer allow you to cross-probe to the source design file and to various other windows in the Quartus Prime software.

You can select one or more hierarchy boxes, nodes, state nodes, or state transition arcs that interest you in the Netlist Viewer and locate the corresponding items in another applicable Quartus Prime software window. You can then view and make changes or assignments in the appropriate editor or floorplan.

To locate an item from the Netlist Viewer in another window, right-click the items of interest in the schematic or state diagram, point to **Locate**, and click the appropriate command. The following commands are available:

- **Locate in Assignment Editor**
- **Locate in Pin Planner**
- **Locate in Chip Planner**
- **Locate in Resource Property Editor**
- **Locate in Technology Map Viewer**
- **Locate in RTL Viewer**
- **Locate in Design File**

The options available for locating an item depend on the type of node and whether it exists after placement and routing. If a command is enabled in the menu, it is available for the selected node. You can use the **Locate in Assignment Editor** command for all nodes, but assignments might be ignored during placement and routing if they are applied to nodes that do not exist after synthesis.

The Netlist Viewer automatically opens another window for the appropriate editor or floorplan and highlights the selected node or net in the newly opened window. You can switch back to the Netlist Viewer by selecting it in the Window menu or by closing, minimizing, or moving the new window.

17.8 Cross-Probing to the Netlist Viewers from Other Quartus Prime Windows

You can cross-probe to the RTL Viewer and Technology Map Viewer from other windows in the Quartus Prime software. You can select one or more nodes or nets in another window and locate them in one of the Netlist Viewers.

You can locate nodes between the RTL Viewer and Technology Map Viewer, and you can locate nodes in the RTL Viewer and Technology Map Viewer from the following Quartus Prime software windows:

- Project Navigator
- Timing Closure Floorplan
- Chip Planner
- Resource Property Editor
- Node Finder
- Assignment Editor
- Messages Window
- Compilation Report
- TimeQuest Timing Analyzer (supports the Technology Map Viewer only)

To locate elements in the Netlist Viewer from another Quartus Prime window, select the node or nodes in the appropriate window; for example, select an entity in the **Entity** list on the **Hierarchy** tab in the Project Navigator, or select nodes in the Timing Closure Floorplan, or select node names in the **From** or **To** column in the Assignment Editor. Next, right-click the selected object, point to **Locate**, and click **Locate in RTL Viewer** or **Locate in Technology Map Viewer**. After you click this command, the Netlist Viewer opens, or is brought to the foreground if the Netlist Viewer is open.

Note: The first time the window opens after a compilation, the preprocessor stage runs before the Netlist Viewer opens.

The Netlist Viewer shows the selected nodes and, if applicable, the connections between the nodes. The display is similar to what you see if you right-click the object, then click **Filter ➤ Selected Nodes** using **Filter across hierarchy**. If the nodes cannot be found in the Netlist Viewer, a message box displays the message: **Can't find requested location**.

17.9 Viewing a Timing Path

You can cross-probe from a report panel in the TimeQuest Timing Analyzer to see a visual representation of a timing path.

To take advantage of this feature, you must complete a full compilation of your design, including the timing analyzer stage. To see the timing results for your design, on the Processing menu, click **Compilation Report**. On the left side of the Compilation Report, select **TimeQuest Timing Analyzer**. When you select a detailed report, the timing information is listed in a table format on the right side of the Compilation Report; each row of the table represents a timing path in the design. You can also view timing paths in TimeQuest analyzer report panels. To view a particular timing path in the Technology Map Viewer or RTL Viewer, right-click the appropriate row in the table, point to **Locate**, and click **Locate in Technology Map Viewer** or **Locate in RTL Viewer**.

- To locate a path, on the **Tasks** pane click**Custom Reports ➤ Report Timing**.
- In the **Report Timing** dialog box, make necessary settings, and then click the **Report Timing** button.
- After the TimeQuest analyzer generates the report, right-click on the node in the table and select **Locate Path**. In the Technology Map Viewer, the schematic page displays the nodes along the timing path with a summary of the total delay.

When you locate the timing path from the TimeQuest analyzer to the Technology Map Viewer, the interconnect and cell delay associated with each node is displayed on top of the schematic symbols. The total slack of the selected timing path is displayed in the Page Title section of the schematic.

In the RTL Viewer, the schematic page displays the nodes in the paths between the source and destination registers with a summary of the total delay.

The RTL Viewer netlist is based on an initial stage of synthesis, so the post-fitting nodes might not exist in the RTL Viewer netlist. Therefore, the internal delay numbers are not displayed in the RTL Viewer as they are in the Technology Map Viewer, and the timing path might not be displayed exactly as it appears in the timing analysis report. If multiple paths exist between the source and destination registers, the RTL Viewer might display more than just the timing path. There are also some cases in which the

path cannot be displayed, such as paths through state machines, encrypted intellectual property (IP), or registers that are created during the fitting process. In cases where the timing path displayed in the RTL Viewer might not be the correct path, the compiler issues messages.

17.10 Document Revision History

Date	Version	Changes
2016.10.31	16.1.0	• Implemented Intel rebranding.
2016.05.03	16.0.0	Removed Schematic Viewer topic.
2015.11.02	15.1.0	Added Schematic Viewer topic for viewing stage snapshots. Added information for the following new features and feature updatess: • Nets visible across hierarchies • Connection Details • Display Settings • Hand Tool • Area Selection Tool • New default behavior for Show/Hide Instance Pins (default is now off)
2014.06.30	14.0.0	Added Show Netlist on One Page and show/Hide Instance Pins commands.
November 2013	13.1.0	Removed HardCopy device information. Reorganized and migrated to new template. Added support for new Netlist viewer.
November 2012	12.1.0	Added sections to support Global Net Routing feature.
June 2012	12.0.0	Removed survey link.
November 2011	10.0.2	Template update.
December 2010	10.0.1	Changed to new document template.
July 2010	10.0.0	• Updated screenshots • Updated chapter for the Quartus Prime software version 10.0, including major user interface changes
November 2009	9.1.0	• Updated devices • Minor text edits
March 2009	9.0.0	• Chapter 13 was formerly Chapter 12 in version 8.1.0 • Updated Figure 13–2, Figure 13–3, Figure 13–4, Figure 13–14, and Figure 13–30 • Added "Enable or Disable the Auto Hierarchy List" on page 13–15 • Updated "Find Command" on page 13–44
November 2008	8.1.0	Changed page size to 8.5" × 11"
May 2008	8.0.0	• Added Arria GX support • Updated operator symbols • Updated information about the radial menu feature • Updated zooming feature • Updated information about probing from schematic to Signal Tap Analyzer • Updated constant signal information • Added .png and .gif to the list of supported image file formats • Updated several figures and tables

continued...

Date	Version	Changes
		• Added new sections "Enabling and Disabling the Radial Menu", "Changing the Time Interval", "Changing the Constant Signal Value Formatting", "Logic Clouds in the RTL Viewer", "Logic Clouds in the Technology Map Viewer", "Manually Group and Ungroup Logic Clouds", "Customizing the Shortcut Commands" • Renamed several sections • Removed section "Customizing the Radial Menu" • Moved section "Grouping Combinational Logic into Logic Clouds" • Updated document content based on the Quartus Prime software version 8.0

Related Links

Altera Documentation Archive

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

18 Mentor Graphics Precision Synthesis Support

18.1 About Precision RTL Synthesis Support

This manual delineates the support for the Mentor Graphics® Precision RTL Synthesis and Precision RTL Plus Synthesis software in the Quartus Prime software, as well as key design flows, methodologies and techniques for improving your results for Intel devices. This manual assumes that you have set up, licensed, and installed the Precision Synthesis software and the Quartus Prime software.

To obtain and license the Precision Synthesis software, refer to the Mentor Graphics website. To install and run the Precision Synthesis software and to set up your work environment, refer to the *Precision Synthesis Installation Guide* in the Precision Manuals Bookcase. To access the Manuals Bookcase in the Precision Synthesis software, click **Help** and select **Open Manuals Bookcase**.

Related Links

Mentor Graphics website

18.2 Design Flow

The following steps describe a basic Quartus Prime design flow using the Precision Synthesis software:

1. Create Verilog HDL or VHDL design files.
2. Create a project in the Precision Synthesis software that contains the HDL files for your design, select your target device, and set global constraints.
3. Compile the project in the Precision Synthesis software.
4. Add specific timing constraints, optimization attributes, and compiler directives to optimize the design during synthesis. With the design analysis and cross-probing capabilities of the Precision Synthesis software, you can identify and improve circuit area and performance issues using prelayout timing estimates.

   Note: For best results, Mentor Graphics recommends specifying constraints that are as close as possible to actual operating requirements. Properly setting clock and I/O constraints, assigning clock domains, and indicating false and multicycle paths guide the synthesis algorithms more accurately toward a suitable solution in the shortest synthesis time.
5. Synthesize the project in the Precision Synthesis software.
6. Create a Quartus Prime project and import the following files generated by the Precision Synthesis software into the Quartus Prime project:

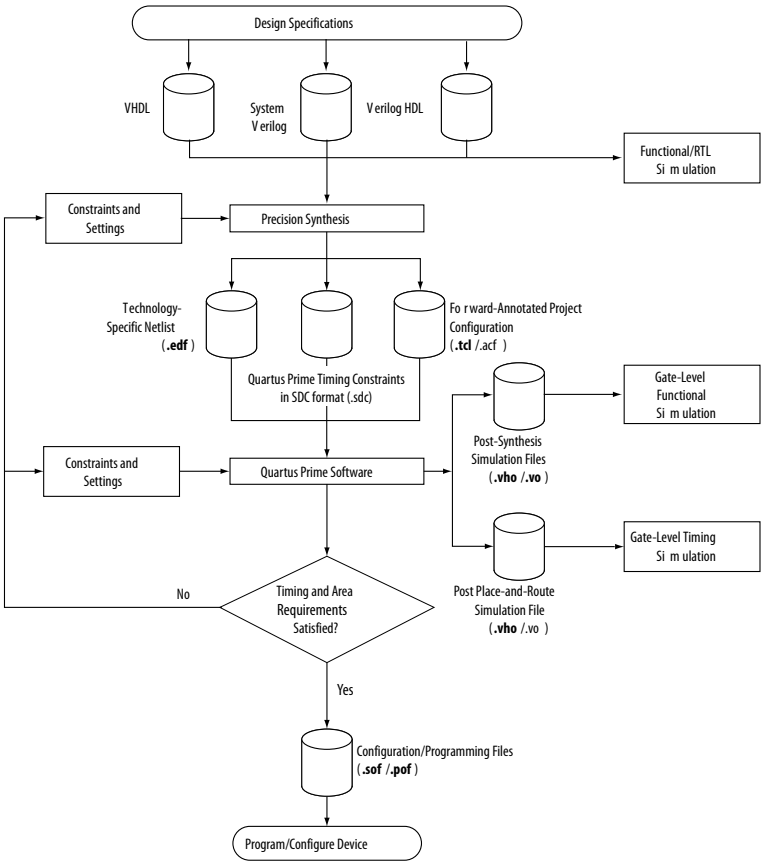
- The Verilog Quartus Mapping File (`.vqm`) netlist
- Synopsys Design Constraints File (`.sdc`) for TimeQuest Timing Analyzer constraints
- Tcl Script Files (`.tcl`) to set up your Quartus Prime project and pass constraints

Note: If your design uses the Classic Timing Analyzer for timing analysis in the Quartus Prime software versions 10.0 and earlier, the Precision Synthesis software generates timing constraints in the Tcl Constraints File (`.tcl`). If you are using the Quartus Prime software versions 10.1 and later, you must use the TimeQuest Timing Analyzer for timing analysis.

7. After obtaining place-and-route results that meet your requirements, configure or program the Intel device.

You can run the Quartus Prime software from within the Precision Synthesis software, or run the Precision Synthesis software using the Quartus Prime software.

Figure 301. Design Flow Using the Precision Synthesis Software and Quartus Prime Software



18.2.1 Timing Optimization

If your area or timing requirements are not met, you can change the constraints and resynthesize the design in the Precision Synthesis software, or you can change the constraints to optimize the design during place-and-route in the Quartus Prime software. Repeat the process until the area and timing requirements are met.

You can use other options and techniques in the Quartus Prime software to meet area and timing requirements. For example, the **WYSIWYG Primitive Resynthesis** option can perform optimizations on your EDIF netlist in the Quartus Prime software.

While simulation and analysis can be performed at various points in the design process, final timing analysis should be performed after placement and routing is complete.

Related Links

- Netlist Optimizations and Physical Synthesis documentation
- Timing Closure and Optimization documentation

18.3 Intel Device Family Support

The Precision Synthesis software supports active devices available in the current version of the Quartus Prime software. Support for newly released device families may require an overlay. Contact Mentor Graphics for more information.

18.4 Precision Synthesis Generated Files

During synthesis, the Precision Synthesis software produces several intermediate and output files.

Table 262. Precision Synthesis Software Intermediate and Output Files

File Extension	File Description
`.psp`	Precision Synthesis Project File.
`.xdb`	Mentor Graphics Design Database File.
`.rep`[9]	Synthesis Area and Timing Report File.
`.vqm`[10]	Technology-specific netlist in `.vqm` file format. By default, the Precision Synthesis software creates `.vqm` files for Arria series, Cyclone series, and Stratix series devices. The Precision Synthesis software defaults to creating `.vqm` files when the device is supported.

continued...

9 The timing report file includes performance estimates that are based on pre-place-and-route information. Use the f_{MAX} reported by the Quartus Prime software after place-and-route for accurate post-place-and-route timing information. The area report file includes post-synthesis device resource utilization statistics that can differ from the resource usage after place-and-route due to black boxes or further optimizations performed during placement and routing. Use the device utilization reported by the Quartus Prime software after place-and-route for final resource utilization results.

10 The Precision Synthesis software-generated VQM file is supported by the Quartus Prime software version 10.1 and later.

File Extension	File Description
.tcl	Forward-annotated Tcl assignments and constraints file. The <*project name*>.tcl file is generated for all devices. The .tcl file acts as the Quartus Prime Project Configuration file and is used to make basic project and placement assignments, and to create and compile a Quartus Prime project.
.acf	Assignment and Configurations file for backward compatibility with the MAX+PLUS II software. For devices supported by the MAX+PLUS II software, the MAX+PLUS II assignments are imported from the MAX+PLUS II .acf file.
.sdc	Quartus Prime timing constraints file in Synopsys Design Constraints format. This file is generated automatically if the device uses the TimeQuest Timing Analyzer by default in the Quartus Prime software, and has the naming convention <*project name*>_pnr_constraints .sdc.

Related Links

Synthesizing the Design and Evaluating the Results on page 1011
During synthesis, the Precision Synthesis software optimizes the compiled design, and then writes out netlists and reports to the implementation subdirectory of your working directory after the implementation is saved, using the following naming convention:

18.5 Creating and Compiling a Project in the Precision Synthesis Software

After creating your design files, create a project in the Precision Synthesis software that contains the basic settings for compiling the design.

18.6 Mapping the Precision Synthesis Design

In the next steps, you set constraints and map the design to technology-specific cells. The Precision Synthesis software maps the design by default to the fastest possible implementation that meets your timing constraints. To accomplish this, you must specify timing requirements for the automatically determined clock sources. With this information, the Precision Synthesis software performs static timing analysis to determine the location of the critical timing paths. The Precision Synthesis software achieves the best results for your design when you set as many realistic constraints as possible. Be sure to set constraints for timing, mapping, false paths, multicycle paths, and other factors that control the structure of the implemented design.

Mentor Graphics recommends creating an .**sdc** file and adding this file to the **Constraint Files** section of the **Project Files** list. You can create this file with a text editor, by issuing command-line constraint parameters, or by directing the Precision Synthesis software to generate the file automatically the first time you synthesize your design. By default, the Precision Synthesis software saves all timing constraints and attributes in two files: **precision_rtl.sdc** and **precision_tech.sdc**. The **precision_rtl.sdc** file contains constraints set on the RTL-level database (post-compilation) and the **precision_tech.sdc** file contains constraints set on the gate-level database (post- synthesis) located in the current implementation directory.

You can also enter constraints at the command line. After adding constraints at the command line, update the .**sdc** file with the `update constraint file` command. You can add constraints that change infrequently directly to the HDL source files with HDL attributes or pragmas.

Note: The Precision .**sdc** file contains all the constraints for the Precision Synthesis project. For the Quartus Prime software, placement constraints are written in a .**tcl** file and timing constraints for the TimeQuest Timing Analyzer are written in the Quartus Prime .**sdc** file.

For details about the syntax of Synopsys Design Constraint commands, refer to the *Precision RTL Synthesis User's Manual* and the *Precision Synthesis Reference Manual*. For more details and examples of attributes, refer to the *Attributes* chapter in the *Precision Synthesis Reference Manual.*

18.6.1 Setting Timing Constraints

The Precision Synthesis software uses timing constraints, based on the industry-standard .**sdc** file format, to deliver optimal results. Missing timing constraints can result in incomplete timing analysis and might prevent timing errors from being detected. The Precision Synthesis software provides constraint analysis prior to synthesis to ensure that designs are fully and accurately constrained. The <*project name*>**_pnr_constraints.sdc** file, which contains timing constraints in SDC format, is generated in the Quartus Prime software.

Note: Because the .**sdc** file format requires that timing constraints be set relative to defined clocks, you must specify your clock constraints before applying any other timing constraints.

You also can use multicycle path and false path assignments to relax requirements or exclude nodes from timing requirements, which can improve area utilization and allow the software optimizations to focus on the most critical parts of the design.

For details about the syntax of Synopsys Design Constraint commands, refer to the *Precision RTL Synthesis User's Manual* and the *Precision Synthesis Reference Manual*.

18.6.2 Setting Mapping Constraints

Mapping constraints affect how your design is mapped into the target Intel device. You can set mapping constraints in the user interface, in HDL code, or with the `set_attribute` command in the constraint file.

18.6.3 Assigning Pin Numbers and I/O Settings

The Precision Synthesis software supports assigning device pin numbers, I/O standards, drive strengths, and slew-rate settings to top-level ports of the design. You can set these timing constraints with the `set_attribute` command, the GUI, or by specifying synthesis attributes in your HDL code. These constraints are forward-annotated in the <*project name*>.**tcl** file that is read by the Quartus Prime software during place-and-route and do not affect synthesis.

You can use the `set_attribute` command in the Precision Synthesis software .**sdc** file to specify pin number constraints, I/O standards, drive strengths, and slow slew-rate settings. The table below describes the format to use for entries in the Precision Synthesis software constraint file.

Table 263. Constraint File Settings

Constraint	Entry Format for Precision Constraint File
Pin number	`set_attribute -name PIN_NUMBER -value "<pin number>" -port <port name>`
I/O standard	`set_attribute -name IOSTANDARD -value "<I/O Standard>" -port <port name>`
Drive strength	`set_attribute -name DRIVE -value "<drive strength in mA>" -port <port name>`
Slew rate	`set_attribute -name SLEW -value "TRUE | FALSE" -port <port name>`

You also can use synthesis attributes or pragmas in your HDL code to make these assignments.

Example 104. Verilog HDL Pin Assignment

```
//pragma attribute clk pin_number P10;
```

Example 105. VHDL Pin Assignment

```
attribute pin_number : string
attribute pin_number of clk : signal is "P10";
```

You can use the same syntax to assign the I/O standard using the IOSTANDARD attribute, drive strength using the attribute DRIVE, and slew rate using the SLEW attribute.

For more details about attributes and how to set these attributes in your HDL code, refer to the *Precision Synthesis Reference Manual.*

18.6.4 Assigning I/O Registers

The Precision Synthesis software performs timing-driven I/O register mapping by default. You can force a register to the device IO element (IOE) using the Complex I/O constraint. This option does not apply if you turn off **I/O pad insertion.**

Note: You also can make the assignment by right-clicking on the pin in the Schematic Viewer.

For the Stratix series, Cyclone series, and the MAX II device families, the Precision Synthesis software can move an internal register to an I/O register without any restrictions on design hierarchy.

For more mature devices, the Precision Synthesis software can move an internal register to an I/O register only when the register exists in the top-level of the hierarchy. If the register is buried in the hierarchy, you must flatten the hierarchy so that the buried registers are moved to the top-level of the design.

18.6.5 Disabling I/O Pad Insertion

The Precision Synthesis software assigns I/O pad atoms (device primitives used to represent the I/O pins and I/O registers) to all ports in the top-level of a design by default. In certain situations, you might not want the software to add I/O pads to all I/O pins in the design. The Quartus Prime software can compile a design without I/O pads; however, including I/O pads provides the Precision Synthesis software with more information about the top-level pins in the design.

18.6.5.1 Preventing the Precision Synthesis Software from Adding I/O Pads

If you are compiling a subdesign as a separate project, I/O pins cannot be primary inputs or outputs of the device; therefore, the I/O pins should not have an I/O pad associated with them.

To prevent the Precision Synthesis software from adding I/O pads:

- You can use the Precision Synthesis GUI or add the following command to the project file:

```
setup_design -addio=false
```

18.6.5.2 Preventing the Precision Synthesis Software from Adding an I/O Pad on an Individual Pin

To prevent I/O pad insertion on an individual pin when you are using a black box, such as DDR or a phase-locked loop (PLL), at the external ports of the design, perform the following steps:

1. Compile your design.
2. Use the Precision Synthesis GUI to select the individual pin and turn off I/O pad insertion.

Note: You also can make this assignment by attaching the `nopad` attribute to the port in the HDL source code.

18.6.6 Controlling Fan-Out on Data Nets

Fan-out is defined as the number of nodes driven by an instance or top-level port. High fan-out nets can cause significant delays that result in an unroutable net. On a critical path, high fan-out nets can cause longer delays in a single net segment that result in the timing constraints not being met. To prevent this behavior, each device family has a global fan-out value set in the Precision Synthesis software library. In addition, the Quartus Prime software automatically routes high fan-out signals on global routing lines in the Intel device whenever possible.

To eliminate routability and timing issues associated with high fan-out nets, the Precision Synthesis software also allows you to override the library default value on a global or individual net basis. You can override the library value by setting a `max_fanout` attribute on the net.

18.7 Synthesizing the Design and Evaluating the Results

During synthesis, the Precision Synthesis software optimizes the compiled design, and then writes out netlists and reports to the implementation subdirectory of your working directory after the implementation is saved, using the following naming convention:

`<project name>_impl_<number>`

After synthesis is complete, you can evaluate the results for area and timing. The *Precision RTL Synthesis User's Manual* describes different results that can be evaluated in the software.

There are several schematic viewers available in the Precision Synthesis software: RTL schematic, Technology-mapped schematic, and Critical Path schematic. These analysis tools allow you to quickly and easily isolate the source of timing or area issues, and to make additional constraint or code changes to optimize the design.

18.7.1 Obtaining Accurate Logic Utilization and Timing Analysis Reports

Historically, designers have relied on post-synthesis logic utilization and timing reports to determine the amount of logic their design requires, the size of the device required, and how fast the design runs. However, today's FPGA devices provide a wide variety of advanced features in addition to basic registers and look-up tables (LUTs). The Quartus Prime software has advanced algorithms to take advantage of these features, as well as optimization techniques to increase performance and reduce the amount of logic required for a given design. In addition, designs can contain black boxes and functions that take advantage of specific device features. Because of these advances, synthesis tool reports provide post-synthesis area and timing estimates, but you should use the place-and-route software to obtain final logic utilization and timing reports.

18.8 Guidelines for Intel FPGA IP Cores and Architecture-Specific Features

Intel provides parameterizable IP cores, including the LPMs, device-specific Intel FPGA IP cores, and IP available through the Altera Megafunction Partners Program (AMPP^{SM}). You can use IP cores by instantiating them in your HDL code or by inferring certain functions from generic HDL code.

If you want to instantiate an IP core such as a PLL in your HDL code, you can instantiate and parameterize the function using the port and parameter definitions, or you can customize a function with the Parameter Editor. Intel recommends using the IP Catalog and Parameter Editor, which provides a graphical interface within the Quartus Prime software for customizing and parameterizing any available IP core for the design.

The Precision Synthesis software automatically recognizes certain types of HDL code and infers the appropriate IP core.

Related Links

- Inferring Intel FPGA IP Cores from HDL Code on page 1014

The Precision Synthesis software automatically recognizes certain types of HDL code and maps arithmetical operators, relational operators, and memory (RAM and ROM), to technology-specific implementations.

- Recommended HDL Coding Styles documentation on page 102
  This chapter provides Hardware Description Language (HDL) coding style recommendations to ensure optimal synthesis results when targeting Intel FPGA devices.
- Introduction to Intel FPGA IP Cores documentation

18.8.1 Instantiating IP Cores With IP Catalog-Generated Verilog HDL Files

The IP Catalog generates a Verilog HDL instantiation template file <*output file*>_**inst.v** and a hollow-body black box module declaration <*output file*>_**bb.v** for use in your Precision Synthesis design. Incorporate the instantiation template file, <*output file*>_**inst.v**, into your top-level design to instantiate the IP core wrapper file, <*output file*>.**v**.

Include the hollow-body black box module declaration <*output file*>_**bb.v** in your Precision Synthesis project to describe the port connections of the black box. Adding the IP core wrapper file <*output file*>.**v** in your Precision Synthesis project is optional, but you must add it to your Quartus Prime project along with the Precision Synthesis-generated EDIF or VQM netlist.

Alternatively, you can include the IP core wrapper file <*output file*>.**v** in your Precision Synthesis project and turn on the **Exclude file from Compile Phase** option in the Precision Synthesis software to exclude the file from compilation and to copy the file to the appropriate directory for use by the Quartus Prime software during place-and-route.

18.8.2 Instantiating IP Cores With IP Catalog-Generated VHDL Files

The IP Catalog generates a VHDL component declaration file <*output file*>.**cmp** and a VHDL instantiation template file <*output file*>_**inst.vhd** for use in your Precision Synthesis design. Incorporate the component declaration and instantiation template into your top-level design to instantiate the IP core wrapper file, <*output file*>.**vhd**.

Adding the IP core wrapper file <*output file*>.**vhd** in your Precision Synthesis project is optional, but you must add the file to your Quartus Prime project along with the Precision Synthesis-generated EDIF or VQM netlist.

Alternatively, you can include the IP core wrapper file <*output file*>.**v** in your Precision Synthesis project and turn on the **Exclude file from Compile Phase** option in the Precision Synthesis software to exclude the file from compilation and to copy the file to the appropriate directory for use by the Quartus Prime software during place-and-route.

18.8.3 Instantiating Intellectual Property With the IP Catalog and Parameter Editor

Many Intel FPGA IP functions include a resource and timing estimation netlist that the Precision Synthesis software can use to synthesize and optimize logic around the IP efficiently. As a result, the Precision Synthesis software provides better timing correlation, area estimates, and Quality of Results (QoR) than a black box approach.

To create this netlist file, perform the following steps:

1. Select the IP function in the IP Catalog.
2. Click **Next** to open the Parameter Editor.
3. Click **Set Up Simulation**, which sets up all the EDA options.
4. Turn on the **Generate netlist** option to generate a netlist for resource and timing estimation and click **OK**.
5. Click **Generate** to generate the netlist file.

The Quartus Prime software generates a file <*output file*>_**syn.v**. This netlist contains the "grey box" information for resource and timing estimation, but does not contain the actual implementation. Include this netlist file into your Precision Synthesis project as an input file. Then include the IP core wrapper file <*output file*>.**v|vhd** in the Quartus Prime project along with your EDIF or VQM output netlist.

The generated "grey box" netlist file, <*output file*>_**syn.v** , is always in Verilog HDL format, even if you select VHDL as the output file format.

Note: For information about creating a grey box netlist file from the command line, search Altera's Knowledge Database.

Related Links

Altera Knowledge Center website

18.8.4 Instantiating Black Box IP Functions With Generated Verilog HDL Files

You can use the `syn_black_box` or `black_box` compiler directives to declare a module as a black box. The top-level design files must contain the IP port mapping and a hollow-body module declaration. You can apply the directive to the module declaration in the top-level file or a separate file included in the project so that the Precision Synthesis software recognizes the module is a black box.

Note: The `syn_black_box` and `black_box` directives are supported only on module or entity definitions.

The example below shows a sample top-level file that instantiates **my_verilogIP.v**, which is a simplified customized variation generated by the IP Catalog and Parameter Editor.

Example 106. Top-Level Verilog HDL Code with Black Box Instantiation of IP

```
module top (clk, count);
   input clk;
   output[7:0] count;

   my_verilogIP verilogIP_inst (.clock (clk), .q (count));
endmodule

// Module declaration
// The following attribute is added to create a
// black box for this module.
module my_verilogIP (clock, q) /* synthesis syn_black_box */;
```

```
   input clock;
   output[7:0] q;
endmodule
```

18.8.5 Instantiating Black Box IP Functions With Generated VHDL Files

You can use the syn_black_box or black_box compiler directives to declare a component as a black box. The top-level design files must contain the IP core variation component declaration and port mapping. Apply the directive to the component declaration in the top-level file.

Note: The syn_black_box and black_box directives are supported only on module or entity definitions.

The example below shows a sample top-level file that instantiates **my_vhdlIP.vhd**, which is a simplified customized variation generated by the IP Catalog and Parameter Editor.

Example 107. Top-Level VHDL Code with Black Box Instantiation of IP

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY top IS
  PORT (
    clk: IN STD_LOGIC ;
    count: OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
  );
END top;

ARCHITECTURE rtl OF top IS
  COMPONENT my_vhdlIP
  PORT (
    clock: IN STD_LOGIC ;
    q: OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
  );
  end COMPONENT;
  attribute syn_black_box : boolean;
  attribute syn_black_box of my_vhdlIP: component is true;
  BEGIN
    vhdlIP_inst : my_vhdlIP PORT MAP (
       clock => clk,
       q => count
    );
END rtl;
```

18.8.6 Inferring Intel FPGA IP Cores from HDL Code

The Precision Synthesis software automatically recognizes certain types of HDL code and maps arithmetical operators, relational operators, and memory (RAM and ROM), to technology-specific implementations. This functionality allows technology-specific resources to implement these structures by inferring the appropriate Intel function to provide optimal results. In some cases, the Precision Synthesis software has options that you can use to disable or control inference.

For coding style recommendations and examples for inferring technology-specific architecture in Intel devices, refer to the *Precision Synthesis Style Guide*.

Related Links

Recommended HDL Coding Styles documentation on page 102
This chapter provides Hardware Description Language (HDL) coding style recommendations to ensure optimal synthesis results when targeting Intel FPGA devices.

18.8.6.1 Multipliers

The Precision Synthesis software detects multipliers in HDL code and maps them directly to device atoms to implement the multiplier in the appropriate type of logic. The Precision Synthesis software also allows you to control the device resources that are used to implement individual multipliers.

18.8.6.1.1 Controlling DSP Block Inference for Multipliers

By default, the Precision Synthesis software uses DSP blocks available in Stratix series devices to implement multipliers. The default setting is **AUTO**, which allows the Precision Synthesis software to map to logic look-up tables (LUTs) or DSP blocks, depending on the size of the multiplier. You can use the Precision Synthesis GUI or HDL attributes for direct mapping to only logic elements or to only DSP blocks.

Table 264. Options for dedicated_mult Parameter to Control Multiplier Implementation in Precision Synthesis

Value	Description
ON	Use only DSP blocks to implement multipliers, regardless of the size of the multiplier.
OFF	Use only logic (LUTs) to implement multipliers, regardless of the size of the multiplier.
AUTO	Use logic (LUTs) or DSP blocks to implement multipliers, depending on the size of the multipliers.

18.8.6.2 Setting the Use Dedicated Multiplier Option

To set the `Use Dedicated Multiplier` option in the Precision Synthesis GUI, compile the design, and then in the Design Hierarchy browser, right-click the operator for the desired multiplier and click **Use Dedicated Multiplier**.

18.8.6.3 Setting the dedicated_mult Attribute

To control the implementation of a multiplier in your HDL code, use the `dedicated_mult` attribute with the appropriate value as shown in the examples below.

Example 108. Setting the dedicated_mult Attribute in Verilog HDL

```
//synthesis attribute <signal name> dedicated_mult <value>
```

Example 109. Setting the dedicated_mult Attribute in VHDL

```
ATTRIBUTE dedicated_mult: STRING;
ATTRIBUTE dedicated_mult OF <signal name>: SIGNAL IS <value>;
```

The `dedicated_mult` attribute can be applied to signals and wires; it does not work when applied to a register. This attribute can be applied only to simple multiplier code, such as `a = b * c`.

Some signals for which the `dedicated_mult` attribute is set can be removed during synthesis by the Precision Synthesis software for design optimization. In such cases, if you want to force the implementation, you should preserve the signal by setting the `preserve_signal` attribute to TRUE.

Example 110. Setting the preserve_signal Attribute in Verilog HDL

```
//synthesis attribute <signal name> preserve_signal TRUE
```

Example 111. Setting the preserve_signal Attribute in VHDL

```
ATTRIBUTE preserve_signal: BOOLEAN;
ATTRIBUTE preserve_signal OF <signal name>: SIGNAL IS TRUE;
```

Example 112. Verilog HDL Multiplier Implemented in Logic

```
module unsigned_mult (result, a, b);
   output [15:0] result;
   input [7:0] a;
   input [7:0} b;
   assign result = a * b;
   //synthesis attribute result dedicated_mult OFF
endmodule
```

Example 113. VHDL Multiplier Implemented in Logic

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;
USE ieee.std_logic_unsigned.ALL;

ENTITY unsigned_mult IS
   PORT(
      a: IN std_logic_vector (7 DOWNTO 0);
      b: IN std_logic_vector (7 DOWNTO 0);
      result: OUT std_logic_vector (15 DOWNTO 0));
ATTRIBUTE dedicated_mult: STRING;
END unsigned_mult;

ARCHITECTURE rtl OF unsigned_mult IS
   SIGNAL a_int, b_int: UNSIGNED (7 downto 0);
   SIGNAL pdt_int: UNSIGNED (15 downto 0);
ATTRIBUTE dedicated_mult OF pdt_int: SIGNAL IS "OFF;
BEGIN
   a_int <= UNSIGNED (a);
   b_int <= UNSIGNED (b);
   pdt_int <= a_int * b_int;
   result <= std_logic_vector(pdt_int);
END rtl;
```

18.8.6.4 Multiplier-Accumulators and Multiplier-Adders

The Precision Synthesis software also allows you to control the device resources used to implement multiply-accumulators or multiply-adders in your project or in a particular module.

The Precision Synthesis software detects multiply-accumulators or multiply-adders in HDL code and infers an ALTMULT_ACCUM or ALTMULT_ADD IP cores so that the logic can be placed in DSP blocks, or the software maps these functions directly to device atoms to implement the multiplier in the appropriate type of logic.

Note: The Precision Synthesis software supports inference for these functions only if the target device family has dedicated DSP blocks.

For more information about DSP blocks in Intel devices, refer to the appropriate Intel device family handbook and device-specific documentation. For details about which functions a given DSP block can implement, refer to the DSP Solutions Center on the Altera website.

For more information about inferring multiply-accumulator and multiply-adder IP cores in HDL code, refer to the Intel *Recommended HDL Coding Styles* and the Mentor Graphics*Precision Synthesis Style Guide*.

Related Links

- Altera DSP Solutions website
- Recommended HDL Coding Styles documentation on page 102
  This chapter provides Hardware Description Language (HDL) coding style recommendations to ensure optimal synthesis results when targeting Intel FPGA devices.

18.8.6.5 Controlling DSP Block Inference

By default, the Precision Synthesis software infers the ALTMULT_ADD or ALTMULT_ACCUM IP cores appropriately in your design. These IP cores allow the Quartus Prime software to select either logic or DSP blocks, depending on the device utilization and the size of the function.

You can use the `extract_mac` attribute to prevent inference of an ALTMULT_ADD or ALTMULT_ACCUM IP cores in a certain module or entity.

Table 265. Options for extract_mac Attribute Controlling DSP Implementation

Value	Description
TRUE	The ALTMULT_ADD or ALTMULT_ACCUM IP core is inferred.
FALSE	The ALTMULT_ADD or ALTMULT_ACCUM IP core is not inferred.

To control inference, use the `extract_mac` attribute with the appropriate value from the examples below in your HDL code.

Example 114. Setting the extract_mac Attribute in Verilog HDL

```
//synthesis attribute <module name> extract_mac <value>
```

Example 115. Setting the extract_mac Attribute in VHDL

```
ATTRIBUTE extract_mac: BOOLEAN;
ATTRIBUTE extract_mac OF <entity name>: ENTITY IS <value>;
```

To control the implementation of the multiplier portion of a multiply-accumulator or multiply-adder, you must use the `dedicated_mult` attribute.

You can use the `extract_mac`, `dedicated_mult`, and `preserve_signal` attributes (in Verilog HDL and VHDL) to implement the given DSP function in logic in the Quartus Prime software.

Example 116. Using extract_mac, dedicated_mult, and preserve_signal in Verilog HDL

```
module unsig_altmult_accuml (dataout, dataa, datab, clk, aclr, clken);
   input [7:0} dataa, datab;
   input clk, aclr, clken;
   output [31:0] dataout;

   reg    [31:0] dataout;
   wire   [15:0] multa;
   wire   [31:0] adder_out;

   assign multa = dataa * datab;

   //synthesis attribute multa preserve_signal TRUE
   //synthesis attribute multa dedicated_mult OFF
   assign adder_out = multa + dataout;

   always @ (posedge clk or posedge aclr)
   begin
      if (aclr)
      dataout <= 0;
      else if (clken)
      dataout <= adder_out;
   end

   //synthesis attribute unsig_altmult_accuml extract_mac FALSE
endmodule
```

Example 117. Using extract_mac, dedicated_mult, and preserve_signal in VHDL

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_signed.all;
ENTITY signedmult_add IS
   PORT(
      a, b, c, d: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
      result: OUT STD_LOGIC_VECTOR (15 DOWNTO 0));
   ATTRIBUTE preserve_signal: BOOLEANS;
   ATTRIBUTE dedicated_mult: STRING;
   ATTRIBUTE extract_mac: BOOLEAN;
   ATTRIBUTE extract_mac OF signedmult_add: ENTITY IS FALSE;
END signedmult_add;
ARCHITECTURE rtl OF signedmult_add IS
   SIGNAL a_int, b_int, c_int, d_int : signed (7 DOWNTO 0);
   SIGNAL pdt_int, pdt2_int : signed (15 DOWNTO 0);
   SIGNAL result_int: signed (15 DOWNTO 0);
   ATTRIBUTE preserve_signal OF pdt_int: SIGNAL IS TRUE;
```

```
   ATTRIBUTE dedicated_mult OF pdt_int: SIGNAL IS "OFF";
   ATTRIBUTE preserve_signal OF pdt2_int: SIGNAL IS TRUE;
   ATTRIBUTE dedicated_mult OF pdt2_int: SIGNAL IS "OFF";
BEGIN
   a_int <= signed (a);
   b_int <= signed (b);
   c_int <= signed (c);
   d_int <= signed (d);
   pdt_int <= a_int * b_int;
   pdt2_int <= c_int * d_int;
   result_int <= pdt_int + pdt2_int;
   result <= STD_LOGIC_VECTOR(result_int);
END rtl;
```

18.8.6.6 RAM and ROM

The Precision Synthesis software detects memory structures in HDL code and converts them to an operator that infers an ALTSYNCRAM or LPM_RAM_DP IP cores, depending on the device family. The software then places these functions in memory blocks.

The software supports inference for these functions only if the target device family has dedicated memory blocks.

For more information about inferring RAM and ROM IP cores in HDL code, refer to the *Precision Synthesis Style Guide*.

Related Links

Recommended HDL Coding Styles documentation on page 102
This chapter provides Hardware Description Language (HDL) coding style recommendations to ensure optimal synthesis results when targeting Intel FPGA devices.

18.9 Document Revision History

Table 266. Document Revision History

Date	Version	Changes
2016.10.31	16.1.0	• Implemented Intel rebranding.
2015.11.02	15.1.0	• Changed instances of *Quartus II* to *Quartus Prime*.
June 2014	14.0.0	• Dita conversion. • Removed obsolete devices. • Replaced Megafunction, MegaWizard, and IP Toolbench content with IP Catalog and Parameter Editor content.
June 2012	12.0.0	• Removed survey link.
November 2011	10.1.1	• Template update. • Minor editorial changes.
December 2010	10.1.0	• Changed to new document template. • Removed Classic Timing Analyzer support. • Added support for **. vqm** netlist files. • Edited the "Creating Quartus Prime Projects for Multiple EDIF Files" on page 15–30 section for changes with the incremental compilation flow. • Editorial changes.
July 2010	10.0.0	• Minor updates for the Quartus Prime software version 10.0 release

continued...

Date	Version	Changes
November 2009	9.1.0	• Minor updates for the Quartus Prime software version 9.1 release
March 2009	9.0.0	• Updated list of supported devices for the Quartus Prime software version 9.0 release • Chapter 11 was previously Chapter 10 in software version 8.1
November 2008	8.1.0	• Changed to 8-1/2 x 11 page size • Title changed to *Mentor Graphics Precision Synthesis Support* • Updated list of supported devices • Added information about the Precision RTL Plus incremental synthesis flow • Updated Figure 10-1 to include SystemVerilog • Updated "Guidelines for Altera Megafunctions and Architecture-Specific Features" on page 10–19 • Updated "Incremental Compilation and Block-Based Design" on page 10–28 • Added section "Creating Partitions with the incr_partition Attribute" on page 10–29
May 2008	8.0.0	• Removed Mercury from the list of supported devices • Changed Precision version to 2007a update 3 • Added note for Stratix IV support • Renamed "Creating a Project and Compiling the Design" section to "Creating and Compiling a Project in the Precision RTL Synthesis Software" • Added information about constraints in the Tcl file • Updated document based on the Quartus Prime software version 8.0

Related Links

Altera Documentation Archive

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

19 Synopsys Synplify Support

19.1 About Synplify Support

the Quartus Prime software supports use of the Synopsys Synplify software design flows, methodologies, and techniques for achieving optimal results in Intel devices.Synplify support applies to Synplify, Synplify Pro, and Synplify Premier software. This document assumes proper set up, licensing, and basic familiarity with the Synplify software.

This document covers the following information:

- General design flow with the Synplify and Quartus Prime software.
- Synplify software optimization strategies, including timing-driven compilation settings, optimization options, and other attributes.
- Guidelines for use of Quartus Prime IP cores, including guidelines for HDL inference of IP cores.

Related Links

- Synplify Synthesis Techniques with the Quartus Prime Software online training
- Synplify Pro Tips and Tricks online training

19.2 Design Flow

The following steps describe a basic Quartus Prime software design flow using the Synplify software:

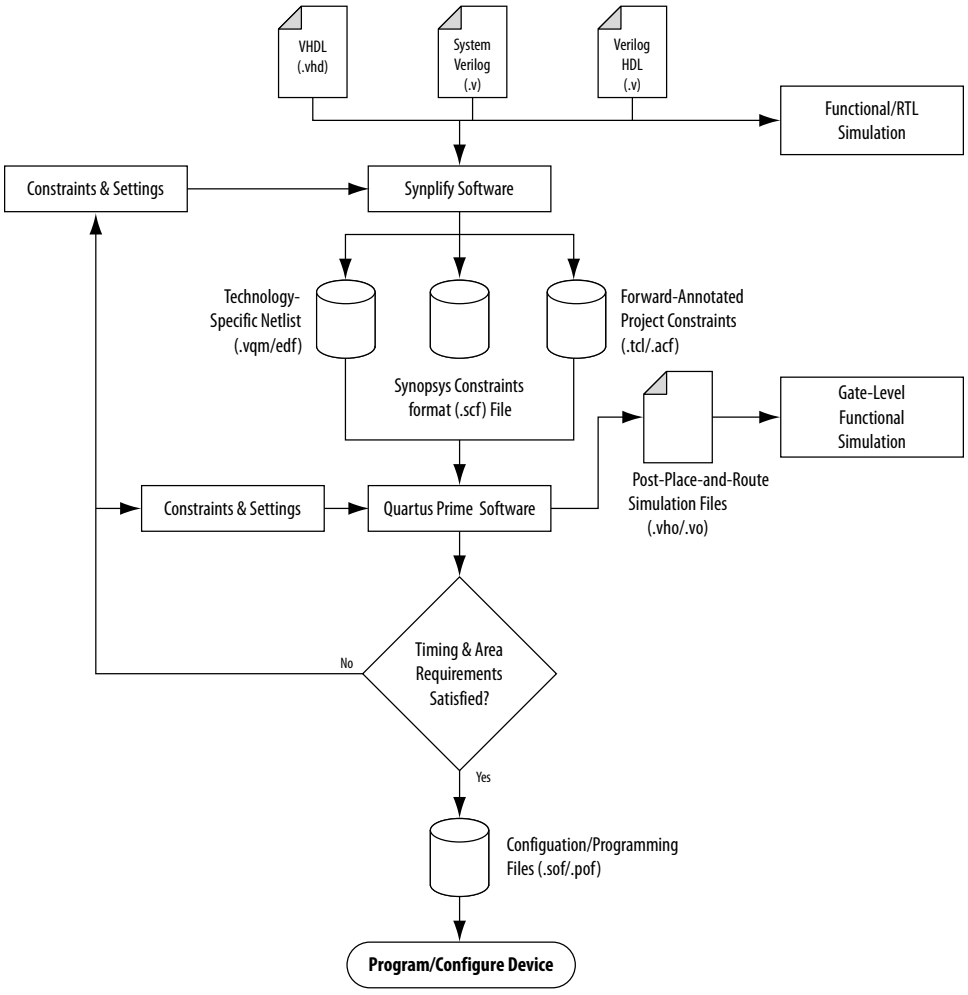
1. Create Verilog HDL (`.v`) or VHDL (`.vhd`) design files.
2. Set up a project in the Synplify software and add the HDL design files for synthesis.
3. Select a target device and add timing constraints and compiler directives in the Synplify software to help optimize the design during synthesis.
4. Synthesize the project in the Synplify software.
5. Create a Quartus Prime project and import the following files generated by the Synplify software into the Quartus Prime software. Use the following files for placement and routing, and for performance evaluation:

- Verilog Quartus Mapping File (**.vqm**) netlist.
- The Synopsys Constraints Format (**.scf**) file for TimeQuest Timing Analyzer constraints.
- The **.tcl** file to set up your Quartus Prime project and pass constraints.

  Note: Alternatively, you can run the Quartus Prime software from within the Synplify software.

6. After obtaining place-and-route results that meet your requirements, configure or program the Intel device.

Figure 302. Recommended Design Flow



Related Links

- Synplify Software Generated Files on page 1024
  During synthesis, the Synplify software produces several intermediate and output files.
- Design Constraints Support on page 1025

You can specify timing constraints and attributes by using the SCOPE window of the Synplify software, by editing the **.sdc** file, or by defining the compiler directives in the HDL source file. The Synplify software forward-annotates many of these constraints to the Quartus Prime software.

19.3 Hardware Description Language Support

The Synplify software supports VHDL, Verilog HDL, and SystemVerilog source files. However, only the Synplify Pro and Premier software support mixed synthesis, allowing a combination of VHDL and Verilog HDL or SystemVerilog format source files.

The HDL Analyst that is included in the Synplify software is a graphical tool for generating schematic views of the technology-independent RTL view netlist (**.srs**) and technology-view netlist (**.srm**) files. You can use the Synplify HDL Analyst to analyze and debug your design visually. The HDL Analyst supports cross-probing between the RTL and Technology views, the HDL source code, the Finite State Machine (FSM) viewer, and between the technology view and the timing report file in the Quartus Prime software. A separate license file is required to enable the HDL Analyst in the Synplify software. The Synplify Pro and Premier software include the HDL Analyst.

Related Links

Guidelines for Intel FPGA IP Cores and Architecture-Specific Features on page 1034
Intel provides parameterizable IP cores, including LPMs, device-specific Intel FPGA IP cores, and IP available through the Altera Megafunction Partners Program (AMPP^{SM}). You can use IP cores by instantiating them in your HDL code, or by inferring certain IP cores from generic HDL code.

19.4 Intel Device Family Support

Support for newly released device families may require an overlay. Contact Synopsys for more information.

Related Links

Synopsys Website

19.5 Tool Setup

19.5.1 Specifying the Quartus Prime Software Version

You can specify your version of the Quartus Prime software in **Implementation Options** in the Synplify software. This option ensures that the netlist is compatible with the software version and supports the newest features. Intel recommends using the latest version of the Quartus Prime software whenever possible. If your Quartus Prime software version is newer than the versions available in the **Quartus Version** list, check if there is a newer version of the Synplify software available that supports the current Quartus Prime software version. Otherwise, select the latest version in the list for the best compatibility.

Note: The **Quartus Version** list is available only after selecting an Intel device.

Example 118. Specifying Quartus Prime Software Version at the Command Line

```
set_option -quartus_version <version number>
```

19.6 Synplify Software Generated Files

During synthesis, the Synplify software produces several intermediate and output files.

Table 267. Synplify Intermediate and Output Files

File Extensions	File Description
.vqm	Technology-specific netlist in **.vqm** file format. A **.vqm** file is created for all Intel device families supported by the Quartus Prime software.
.scf[11]	Synopsys Constraint Format file containing timing constraints for the TimeQuest Timing Analyzer.
.tcl	Forward-annotated constraints file containing constraints and assignments. A **.tcl** file for the Quartus Prime software is created for all devices. The **.tcl** file contains the appropriate Tcl commands to create and set up a Quartus Prime project and pass placement constraints.
.srs	Technology-independent RTL netlist file that can be read only by the Synplify software.
.srm	Technology view netlist file.
.acf	Assignment and Configurations file for backward compatibility with the MAX+PLUS II software. For devices supported by the MAX+PLUS II software, the MAX+PLUS II assignments are imported from the MAX+PLUS II **.acf** file.
.srr[12]	Synthesis Report file.

Related Links

Design Flow on page 1021
The following steps describe a basic Quartus Prime software design flow using the Synplify software:

11 If your design uses the Classic Timing Analyzer for timing analysis in the Quartus Prime software versions 10.0 and earlier, the Synplify software generates timing constraints in the Tcl Constraints File (**.tcl**). If you are using the Quartus Prime software versions 10.1 and later, you must use the TimeQuest Timing Analyzer for timing analysis.

12 This report file includes performance estimates that are often based on pre-place-and-route information. Use the f_{MAX} reported by the Quartus Prime software after place-and-route—it is the only reliable source of timing information. This report file includes post-synthesis device resource utilization statistics that might inaccurately predict resource usage after place-and-route. The Synplify software does not account for black box functions nor for logic usage reduction achieved through register packing performed by the Quartus Prime software. Register packing combines a single register and look-up table (LUT) into a single logic cell, reducing logic cell utilization below the Synplify software estimate. Use the device utilization reported by the Quartus Prime software after place-and-route.

19.7 Design Constraints Support

You can specify timing constraints and attributes by using the SCOPE window of the Synplify software, by editing the **.sdc** file, or by defining the compiler directives in the HDL source file. The Synplify software forward-annotates many of these constraints to the Quartus Prime software.

After synthesis is complete, do the following steps:

1. Import the **.vqm** netlist to the Quartus Prime software for place-and-route.
2. Use the **.tcl** file generated by the Synplify software to forward-annotate your project constraints including device selection. The **.tcl** file calls the generated **.scf** to foward-annotate TimeQuest Timing Analyzer timing constraints.

Related Links

- Design Flow on page 1021
  The following steps describe a basic Quartus Prime software design flow using the Synplify software:
- Synplify Optimization Strategies on page 1027
  Combining Synplify software constraints with VHDL and Verilog HDL coding techniques and Quartus Prime software options can help you obtain the results that you require.
- Netlist Optimizations and Physical Synthesis Documentation

19.7.1 Running the Quartus Prime Software Manually With the Synplify-Generated Tcl Script

You can run the Quartus Prime software with a Synplify-generated Tcl script.

To run the Tcl script to set up your project assignments, perform the following steps:

1. Ensure the **.vqm**, **.scf**, and **.tcl** files are located in the same directory.
2. In the Quartus Prime software, on the View menu, point to and click **Tcl Console**. The Quartus Prime Tcl Console opens.
3. At the Tcl Console command prompt, type the following:

```
source <path>/<project name>_cons.tcl
```

19.7.2 Passing TimeQuest SDC Timing Constraints to the Quartus Prime Software

The TimeQuest Timing Analyzer is a powerful ASIC-style timing analysis tool that validates the timing performance of all logic in your design using an industry standard constraints format, Synopsys Design Constraints (SDC).

The Synplify-generated **.tcl** file contains constraints for the Quartus Prime software, such as the device specification and any location constraints. Timing constraints are forward-annotated in the Synopsys Constraints Format (**.scf**) file.

Note: Synopsys recommends that you modify constraints using the SCOPE constraint editor window, rather than using the generated **.sdc**, **.scf**, or **.tcl** file.

The following list of Synplify constraints are converted to the equivalent Quartus Prime SDC commands and are forward-annotated to the Quartus Prime software in the **.scf** file:

- `define_clock`
- `define_input_delay`
- `define_output_delay`
- `define_multicycle_path`
- `define_false_path`

All Synplify constraints described above are mapped to SDC commands for the TimeQuest Timing Analyzer.

For syntax and arguments for these commands, refer to the applicable topic in this manual or refer to Synplify Help. For a list of corresponding commands in the Quartus Prime software, refer to the Quartus Prime Help.

Related Links

- Timing-Driven Synthesis Settings on page 1028
  The Synplify software supports timing-driven synthesis with user-assigned timing constraints to optimize the performance of the design.
- Quartus Prime TimeQuest Timing Analyzer Documentation

19.7.2.1 Individual Clocks and Frequencies

Specify clock frequencies for individual clocks in the Synplify software with the `define_clock` command. This command is passed to the Quartus Prime software with the `create_clock` command.

19.7.2.2 Input and Output Delay

Specify input delay and output delay constraints in the Synplify software with the `define_input_delay` and `define_output_delay` commands, respectively. These commands are passed to the Quartus Prime software with the `set_input_delay` and `set_output_delay` commands.

19.7.2.3 Multicycle Path

Specify a multicycle path constraint in the Synplify software with the `define_multicycle_path` command. This command is passed to the Quartus Prime software with the `set_multicycle_path` command.

19.7.2.4 False Path

Specify a false path constraint in the Synplify software with the `define_false_path` command. This command is passed to the Quartus Prime software with the `set_false_path` command.

19.8 Simulation and Formal Verification

You can perform simulation and formal verification at various stages in the design process. You can perform final timing analysis after placement and routing is complete.

If area and timing requirements are satisfied, use the files generated by the Quartus Prime software to program or configure the Intel device. If your area or timing requirements are not met, you can change the constraints in the Synplify software or the Quartus Prime software and rerun synthesis. Intel recommends that you provide timing constraints in the Synplify software and any placement constraints in the Quartus Prime software. Repeat the process until area and timing requirements are met.

You can also use other options and techniques in the Quartus Prime software to meet area and timing requirements, such as WYSIWYG Primitive Resynthesis, which can perform optimizations on your **.vqm** netlist within the Quartus Prime software.

Note: In some cases, you might be required to modify the source code if the area and timing requirements cannot be met using options in the Synplify and Quartus Prime software.

19.9 Synplify Optimization Strategies

Combining Synplify software constraints with VHDL and Verilog HDL coding techniques and Quartus Prime software options can help you obtain the results that you require.

For more information about applying attributes, refer to the *Synopsys FPGA Synthesis Reference Manual*.

Related Links

- Design Constraints Support on page 1025
  You can specify timing constraints and attributes by using the SCOPE window of the Synplify software, by editing the **.sdc** file, or by defining the compiler directives in the HDL source file. The Synplify software forward-annotates many of these constraints to the Quartus Prime software.
- Recommended Design Practices Documentation on page 155
  This chapter provides design recommendations for Intel FPGA devices.
- Timing Closure and Optimization Documentation

19.9.1 Using Synplify Premier to Optimize Your Design

Compared to other Synplify products, the Synplify Premier software offers additional physical synthesis optimizations. After typical logic synthesis, the Synplify Premier software places and routes the design and attempts to restructure the netlist based on the physical location of the logic in the Intel device. The Synplify Premier software forward-annotates the design netlist to the Quartus Prime software to perform the final placement and routing. In the default flow, the Synplify Premier software also forward-annotates placement information for the critical path(s) in the design, which can improve the compilation time in the Quartus Prime software.

The physical location annotation file is called *<design name>***_plc.tcl**. If you open the Quartus Prime software from the Synplify Premier software user interface, the Quartus Prime software automatically uses this file for the placement information.

The Physical Analyst allows you to examine the placed netlist from the Synplify Premier software, which is similar to the HDL Analyst for a logical netlist. You can use this display to analyze and diagnose potential problems.

19.9.2 Using Implementations in Synplify Pro or Premier

You can create different synthesis results without overwriting the existing results, in the Synplify Pro or Premier software, by creating a new implementation from the Project menu. For each implementation, specify the target device, synthesis options, and constraint files. Each implementation generates its own subdirectory that contains all the resulting files, including `.vqm`, `.scf`, and `.tcl` files, from a compilation of the particular implementation. You can then compare the results of the different implementations to find the optimal set of synthesis options and constraints for a design.

19.9.3 Timing-Driven Synthesis Settings

The Synplify software supports timing-driven synthesis with user-assigned timing constraints to optimize the performance of the design.

The Quartus Prime NativeLink feature allows timing constraints that are applied in the Synplify software to be forward-annotated for the Quartus Prime software with an **.scf** file for timing-driven place and route.

The Synplify Synthesis Report File (**.srr**) contains timing reports of estimated place-and-route delays. The Quartus Prime software can perform further optimizations on a post-synthesis netlist from third-party synthesis tools. In addition, designs might contain black boxes or intellectual property (IP) functions that have not been optimized by the third-party synthesis software. Actual timing results are obtained only after the design has been fully placed and routed in the Quartus Prime software. For these reasons, the Quartus Prime post place-and-route timing reports provide a more accurate representation of the design. Use the statistics in these reports to evaluate design performance.

Related Links

Passing TimeQuest SDC Timing Constraints to the Quartus Prime Software on page 1025

The TimeQuest Timing Analyzer is a powerful ASIC-style timing analysis tool that validates the timing performance of all logic in your design using an industry standard constraints format, Synopsys Design Constraints (SDC).

19.9.3.1 Clock Frequencies

For single-clock designs, you can specify a global frequency when using the push-button flow. While this flow is simple and provides good results, it often does not meet the performance requirements for more advanced designs. You can use timing constraints, compiler directives, and other attributes to help optimize the performance of a design. You can enter these attributes and directives directly in the HDL code. Alternatively, you can enter attributes (not directives) into an **.sdc** file with the SCOPE window in the Synplify software.

Use the SCOPE window to set global frequency requirements for the entire design and individual clock settings. Use the **Clocks** tab in the SCOPE window to specify frequency (or period), rise times, fall times, duty cycle, and other settings. Assigning

individual clock settings, rather than over-constraining the global frequency, helps the Quartus Prime software and the Synplify software achieve the fastest clock frequency for the overall design. The `define_clock` attribute assigns clock constraints.

19.9.3.2 Multiple Clock Domains

The Synplify software can perform timing analysis on unrelated clock domains. Each clock group is a different clock domain and is treated as unrelated to the clocks in all other clock groups. All clocks in a single clock group are assumed to be related, and the Synplify software automatically calculates the relationship between the clocks. You can assign clocks to a new clock group or put related clocks in the same clock group with the **Clocks** tab in the SCOPE window, or with the `define_clock` attribute.

19.9.3.3 Input and Output Delays

Specify the input and output delays for the ports of a design in the **Input/Output** tab of the SCOPE window, or with the `define_input_delay` and `define_output_delay` attributes. The Synplify software does not allow you to assign the t_{CO} and t_{SU} values directly to inputs and outputs. However, a t_{CO} value can be inferred by setting an external output delay; a t_{SU} value can be inferred by setting an external input delay.

Relationship Between t_{CO} and the Output Delay
t_{CO} = clock period – external output delay

Relationship Between t_{SU} and the Input Delay
t_{SU} = clock period – external input delay

When the `syn_forward_io_constraints` attribute is set to 1, the Synplify software passes the external input and output delays to the Quartus Prime software using NativeLink integration. The Quartus Prime software then uses the external delays to calculate the maximum system frequency.

19.9.3.4 Multicycle Paths

A multicycle path is a path that requires more than one clock cycle to propagate. Specify any multicycle paths in the design in the **Multi-Cycle Paths** tab of the SCOPE window, or with the `define_multicycle_path` attribute. You should specify which paths are multicycle to prevent the Quartus Prime and the Synplify compilers from working excessively on a non-critical path. Not specifying these paths can also result in an inaccurate critical path reported during timing analysis.

19.9.3.5 False Paths

False paths are paths that should be ignored during timing analysis, or should be assigned low (or no) priority during optimization. Some examples of false paths include slow asynchronous resets, and test logic that has been added to the design. Set these paths in the **False Paths** tab of the SCOPE window, or use the `define_false_path` attribute.

19.9.4 FSM Compiler

If the FSM Compiler is turned on, the compiler automatically detects state machines in a design, which are then extracted and optimized. The FSM Compiler analyzes state machines and implements sequential, gray, or one-hot encoding, based on the number of states. The compiler also performs unused-state analysis, optimization of unreachable states, and minimization of transition logic. Implementation is based on the number of states, regardless of the coding style in the HDL code.

If the FSM Compiler is turned off, the compiler does not optimize logic as state machines. The state machines are implemented as HDL code. Thus, if the coding style for a state machine is sequential, the implementation is also sequential.

Use the `syn_state_machine` compiler directive to specify or prevent a state machine from being extracted and optimized. To override the default encoding of the FSM Compiler, use the `syn_encoding` directive.

Table 268. `syn_encoding` Directive Values

Value	Description
Sequential	Generates state machines with the fewest possible flipflops. Sequential, also called binary, state machines are useful for area-critical designs when timing is not the primary concern.
Gray	Generates state machines where only one flipflop changes during each transition. Gray-encoded state machines tend to be glitches.
One-hot	Generates state machines containing one flipflop for each state. One-hot state machines typically provide the best performance and shortest clock-to-output delays. However, one-hot implementations are usually larger than sequential implementations.
Safe	Generates extra control logic to force the state machine to the reset state if an invalid state is reached. You can use the safe value in conjunction with any of the other three values, which results in the state machine being implemented with the requested encoding scheme and the generation of the reset logic.

Example 119. Sample VHDL Code for Applying `syn_encoding` Directive

```
SIGNAL current_state : STD_LOGIC_VECTOR (7 DOWNTO 0);
ATTRIBUTE syn_encoding : STRING;
ATTRIBUTE syn_encoding OF current_state : SIGNAL IS "sequential";
```

By default, the state machine logic is optimized for speed and area, which may be potentially undesirable for critical systems. The safe value generates extra control logic to force the state machine to the reset state if an invalid state is reached.

19.9.4.1 FSM Explorer in Synplify Pro and Premier

The Synplify Pro and Premier software use the FSM Explorer to explore different encoding styles for a state machine automatically, and then implement the best encoding based on the overall design constraints. The FSM Explorer uses the FSM Compiler to identify and extract state machines from a design. However, unlike the FSM Compiler, which chooses the encoding style based on the number of states, the FSM Explorer attempts several different encoding styles before choosing a specific one. The trade-off is that the compilation requires more time to analyze the state machine, but finds an optimal encoding scheme for the state machine.

19.9.5 Optimization Attributes and Options

19.9.5.1 Retiming in Synplify Pro and Premier

The Synplify Pro and Premier software can retime a design, which can improve the timing performance of sequential circuits by moving registers (register balancing) across combinational elements. Be aware that retimed registers incur name changes. You can retime your design from **Implementation Options** or you can use the `syn_allow_retiming` attribute.

19.9.5.2 Maximum Fan-Out

When your design has critical path nets with high fan-out, use the `syn_maxfan` attribute to control the fan-out of the net. Setting this attribute for a specific net results in the replication of the driver of the net to reduce overall fan-out. The `syn_maxfan` attribute takes an integer value and applies it to inputs or registers. The `syn_maxfan` attribute cannot be used to duplicate control signals. The minimum allowed value of the attribute is 4. Using this attribute might result in increased logic resource utilization, thus straining routing resources, which can lead to long compilation times and difficult fitting.

If you must duplicate an output register or an output enable register, you can create a register for each output pin by using the `syn_useioff` attribute.

19.9.5.3 Preserving Nets

During synthesis, the compiler maintains ports, registers, and instantiated components. However, some nets cannot be maintained to create an optimized circuit. Applying the `syn_keep` directive overrides the optimization of the compiler and preserves the net during synthesis. The `syn_keep` directive is a Boolean data type value and can be applied to wires (Verilog HDL) and signals (VHDL). Setting the value to **true** preserves the net through synthesis.

19.9.5.4 Register Packing

Intel devices allow register packing into I/O cells. Intel recommends allowing the Quartus Prime software to make the I/O register assignments. However, you can control register packing with the `syn_useioff` attribute. The `syn_useioff` attribute is a Boolean data type value that can be applied to ports or entire modules. Setting the value to **1** instructs the compiler to pack the register into an I/O cell. Setting the value to **0** prevents register packing in both the Synplify and Quartus Prime software.

19.9.5.5 Resource Sharing

The Synplify software uses resource sharing techniques during synthesis, by default, to reduce area. Turning off the **Resource Sharing** option on the **Options** tab of the **Implementation Options** dialog box improves performance results for some designs. You can also turn off the option for a specific module with the `syn_sharing` attribute. If you turn off this option, be sure to check the results to verify improvement in timing performance. If there is no improvement, turn on **Resource Sharing**.

19.9.5.6 Preserving Hierarchy

The Synplify software performs cross-boundary optimization by default, which causes the design to flatten to allow optimization. You can use the `syn_hier` attribute to override the default compiler settings. The `syn_hier` attribute applies a string value to modules, architectures, or both. Setting the value to **hard** maintains the boundaries of a module, architecture, or both, but allows constant propagation. Setting the value to **locked** prevents all cross-boundary optimizations. Use the **locked** setting with the partition setting to create separate design blocks and multiple output netlists.

By default, the Synplify software generates a hierarchical **.vqm** file. To flatten the file, set the `syn_netlist_hierarchy` attribute to **0**.

19.9.5.7 Register Input and Output Delays

Two advanced options, `define_reg_input_delay` and `define_reg_output_delay`, can speed up paths feeding a register, or coming from a register, by a specific number of nanoseconds. The Synplify software attempts to meet the global clock frequency goals for a design as well as the individual clock frequency goals (set with the `define_clock` attribute). You can use these attributes to add a delay to paths feeding into or out of registers to further constrain critical paths. You can slow down a path that is too highly optimized by setting this attributes to a negative number.

The `define_reg_input_delay` and `define_reg_output_delay` options are useful to close timing if your design does not meet timing goals, because the routing delay after placement and routing exceeds the delay predicted by the Synplify software. Rerun synthesis using these options, specifying the actual routing delay (from place-and-route results) so that the tool can meet the required clock frequency. Synopsys recommends that for best results, do not make these assignments too aggressively. For example, you can increase the routing delay value, but do not also use the full routing delay from the last compilation.

In the SCOPE constraint window, the registers panel contains the following options:

- **Register**—Specifies the name of the register. If you have initialized a compiled design, select the name from the list.
- **Type**—Specifies whether the delay is an input or output delay.
- **Route**—Shrinks the effective period for the constrained registers by the specified value without affecting the clock period that is forward-annotated to the Quartus Prime software.

Use the following Tcl command syntax to specify an input or output register delay in nanoseconds.

Example 120. Input and Output Register Delay

```
define_reg_input_delay {<register>} -route <delay in ns>
define_reg_output_delay {<register>} -route <delay in ns>
```

19.9.5.8 syn_direct_enable

This attribute controls the assignment of a clock-enable net to the dedicated enable pin of a register. With this attribute, you can direct the Synplify mapper to use a particular net as the only clock enable when the design has multiple clock enable candidates.

To use this attribute as a compiler directive to infer registers with clock enables, enter the `syn_direct_enable` directive in your source code, instead of the SCOPE spreadsheet.

The `syn_direct_enable` data type is Boolean. A value of **1** or **true** enables net assignment to the clock-enable pin. The following is the syntax for Verilog HDL:

```
object /* synthesis syn_direct_enable = 1 */ ;
```

19.9.5.9 I/O Standard

For certain Intel devices, specify the I/O standard type for an I/O pad in the design with the **I/O Standard** panel in the Synplify SCOPE window.

The Synplify SDC syntax for the `define_io_standard` constraint, in which the `delay_type` must be either `input_delay` or `output_delay`.

Example 121. define_io_standard Constraint

```
define_io_standard [-disable|-enable] {<objectName>} -delay_type \
[input_delay|output_delay] <columnTclName>{<value>} [<columnTclName>{<value>}...]
```

For details about supported I/O standards, refer to the *Synopsys FPGA Synthesis Reference Manual.*

19.9.6 Intel-Specific Attributes

You can use the `altera_chip_pin_lc`, `altera_io_powerup`, and `altera_io_opendrain` attributes with specific Intel device features, which are forward-annotated to the Quartus Prime project, and are used during place-and-route.

19.9.6.1 altera_chip_pin_lc

Use the `altera_chip_pin_lc` attribute to make pin assignments. This attribute applies a string value to inputs and outputs. Use the attribute only on the ports of the top-level entity in the design. Do not use this attribute to assign pin locations from entities at lower levels of the design hierarchy.

Note: The `altera_chip_pin_lc` attribute is not supported for any MAX series device.

In the SCOPE window, set the value of the `altera_chip_pin_lc` attribute to a pin number or a list of pin numbers.

You can use VHDL code for making location assignments for supported Intel devices. Pin location assignments for these devices are written to the output **.tcl** file.

Note: The `data_out` signal is a 4-bit signal; `data_out[3]` is assigned to pin 14 and `data_out[0]` is assigned to pin 15.

Example 122. Making Location Assignments in VHDL

```
ENTITY sample (data_in : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
    data_out: OUT STD_LOGIC_VECTOR (3 DOWNTO 0));
ATTRIBUTE altera_chip_pin_lc : STRING;
ATTRIBUTE altera_chip_pin_lc OF data_out : SIGNAL IS "14, 5, 16, 15";
```

19.9.6.2 altera_io_powerup

Use the `altera_io_powerup` attribute to define the power-up value of an I/O register that has no set or reset. This attribute applies a string value (**high**|**low**) to ports with I/O registers. By default, the power-up value of the I/O register is set to **low**.

19.9.6.3 altera_io_opendrain

Use the `altera_io_opendrain` attribute to specify open-drain mode I/O ports. This attribute applies a boolean data type value to outputs or bidirectional ports for devices that support open-drain mode.

19.10 Guidelines for Intel FPGA IP Cores and Architecture-Specific Features

Intel provides parameterizable IP cores, including LPMs, device-specific Intel FPGA IP cores, and IP available through the Altera Megafunction Partners Program (AMPP^{SM}). You can use IP cores by instantiating them in your HDL code, or by inferring certain IP cores from generic HDL code.

You can instantiate an IP core in your HDL code with the IP Catalog and configure the IP core with the Parameter Editor, or instantiate the IP core using the port and parameter definition. The IP Catalog and Parameter Editor provide a graphical interface within the Quartus Prime software to customize any available Intel FPGA IP core for the design.

The Synplify software also automatically recognizes certain types of HDL code, and infers the appropriate Intel FPGA IP core when an IP core provides optimal results. The Synplify software provides options to control inference of certain types of IP cores.

Related Links

- Hardware Description Language Support on page 1023
  The Synplify software supports VHDL, Verilog HDL, and SystemVerilog source files. However, only the Synplify Pro and Premier software support mixed synthesis, allowing a combination of VHDL and Verilog HDL or SystemVerilog format source files.
- Recommended HDL Coding Styles Documentation on page 102
  This chapter provides Hardware Description Language (HDL) coding style recommendations to ensure optimal synthesis results when targeting Intel FPGA devices.
- About the IP Catalog Online Help

19.10.1 Instantiating Intel FPGA IP Cores with the IP Catalog

When you use the IP Catalog and Parameter Editor to set up and configure an IP core, the IP Catalog creates a VHDL or Verilog HDL wrapper file <*output file*>**.v|vhd** that instantiates the IP core.

The Synplify software uses the Quartus Prime timing and resource estimation netlist feature to report more accurate resource utilization and timing performance estimates, and leverages timing-driven optimization, instead of treating the IP core as a "black box." Including the generated IP core variation wrapper file in your Synplify project, gives the Synplify software complete information about the IP core.

Note: There is an option in the Parameter Editor to generate a netlist for resource and timing estimation. This option is not recommended for the Synplify software because the software automatically generates this information in the background without a separate netlist. If you do create a separate netlist <*output file*>**_syn.v** and use that file in your synthesis project, you must also include the <*output file*>**.v|vhd** file in your Quartus Prime project.

Verify that the correct Quartus Prime version is specified in the Synplify software before compiling the generated file to ensure that the software uses the correct library definitions for the IP core. The **Quartus Version** setting must match the version of the Quartus Prime software used to generate the customized IP core.

In addition, ensure that the *QUARTUS_ROOTDIR* environment variable specifies the installation directory location of the correct Quartus Prime version. The Synplify software uses this information to launch the Quartus Prime software in the background. The environment variable setting must match the version of the Quartus Prime software used to generate the customized IP core.

Related Links

Specifying the Quartus Prime Software Version on page 1023
You can specify your version of the Quartus Prime software in **Implementation Options** in the Synplify software.

19.10.1.1 Instantiating Intel FPGA IP Cores with IP Catalog Generated Verilog HDL Files

If you turn on the <*output file*>**_inst.v** option on the Parameter Editor, the IP Catalog generates a Verilog HDL instantiation template file for use in your Synplify design. The instantiation template file, <*output file*>**_inst.v**, helps to instantiate the IP core variation wrapper file, <*output file*>**.v**, in your top-level design. Include the IP core variation wrapper file <*output file*>**.v** in your Synplify project. The Synplify software includes the IP core information in the output **.vqm** netlist file. You do not need to include the generated IP core variation wrapper file in your Quartus Prime project.

19.10.1.2 Instantiating Intel FPGA IP Cores with IP Catalog Generated VHDL Files

If you turn on the <*output file*>**.cmp** and <*output file*>**_inst.vhd** options on the Parameter Editor, the IP catalog generates a VHDL component declaration file and a VHDL instantiation template file for use in your Synplify design. These files can help you instantiate the IP core variation wrapper file, <*output file*>**.vhd**, in your top-level

design. Include the <*output file*>**.vhd** in your Synplify project. The Synplify software includes the IP core information in the output **.vqm** netlist file. You do not need to include the generated IP core variation wrapper file in your Quartus Prime project.

19.10.1.3 Changing Synplify's Default Behavior for Instantiated Intel FPGA IP Cores

By default, the Synplify software automatically opens the Quartus Prime software in the background to generate a resource and timing estimation netlist for IP cores.

You might want to change this behavior to reduce run times in the Synplify software, because generating the netlist files can take several minutes for large designs, or if the Synplify software cannot access your Quartus Prime software installation to generate the files. Changing this behavior might speed up the compilation time in the Synplify software, but the Quality of Results (QoR) might be reduced.

The Synplify software directs the Quartus Prime software to generate information in two ways:

- Some IP cores provide a "clear box" model—the Synplify software fully synthesizes this model and includes the device architecture-specific primitives in the output **.vqm** netlist file.
- Other IP cores provide a "grey box" model—the Synplify software reads the resource information, but the netlist does not contain all the logic functionality.

  Note: You need to turn on **Generate netlist** when using the grey box model. For more information, see the Quartus Prime online help.

For these IP cores, the Synplify software uses the logic information for resource and timing estimation and optimization, and then instantiates the IP core in the output **.vqm** netlist file so the Quartus Prime software can implement the appropriate device primitives. By default, the Synplify software uses the clear box model when available, and otherwise uses the grey box model.

Related Links

- Including Files for Quartus Prime Placement and Routing Only on page 1039
  In the Synplify software, you can add files to your project that are used only during placement and routing in the Quartus Prime software. This can be useful if you have grey or black boxes for Synplify synthesis that require the full design files to be compiled in the Quartus Prime software.
- Synplify Synthesis Techniques with the Quartus Prime Software online training
  Includes more information about design flows using clear box model and grey box model.
- Generating a Netlist for 3rd Party Synthesis Tools online help

19.10.1.4 Instantiating Intellectual Property with the IP Catalog and Parameter Editor

Many Intel FPGA IP cores include a resource and timing estimation netlist that the Synplify software uses to report more accurate resource utilization and timing performance estimates, and leverage timing-driven optimization rather than a black box function.

To create this netlist file, perform the following steps:

1. Select the IP core in the IP Catalog.
2. Click **Next** to open the Parameter Editor.
3. Click **Set Up Simulation**, which sets up all the EDA options.
4. Turn on the **Generate netlist** option to generate a netlist for resource and timing estimation and click **OK**.
5. Click **Generate** to generate the netlist file.

The Quartus Prime software generates a file <*output file*>**_syn.v**. This netlist contains the grey box information for resource and timing estimation, but does not contain the actual implementation. Include this netlist file in your Synplify project. Next, include the IP core variation wrapper file <*output file*>**.v|vhd** in the Quartus Prime project along with your Synplify **.vqm** output netlist.

If your IP core does not include a resource and timing estimation netlist, the Synplify software must treat the IP core as a black box.

Related Links

Including Files for Quartus Prime Placement and Routing Only on page 1039
In the Synplify software, you can add files to your project that are used only during placement and routing in the Quartus Prime software. This can be useful if you have grey or black boxes for Synplify synthesis that require the full design files to be compiled in the Quartus Prime software.

19.10.1.5 Instantiating Black Box IP Cores with Generated Verilog HDL Files

Use the `syn_black_box` compiler directive to declare a module as a black box. The top-level design files must contain the IP port-mapping and a hollow-body module declaration. Apply the `syn_black_box` directive to the module declaration in the top-level file or a separate file included in the project so that the Synplify software recognizes the module is a black box. The software compiles successfully without this directive, but reports an additional warning message. Using this directive allows you to add other directives.

The example shows a top-level file that instantiates **my_verilogIP.v**, which is a simple customized variation generated by the IP Catalog.

Example 123. Sample Top-Level Verilog HDL Code with Black Box Instantiation of IP

```
module top (clk, count);
    input clk;
    output [7:0] count;
    my_verilogIP verilogIP_inst (.clock (clk), .q (count));
endmodule
// Module declaration
// The following attribute is added to create a
// black box for this module.
module my_verilogIP (clock, q) /* synthesis syn_black_box */;
    input clock;
    output [7:0] q;
endmodule
```

19.10.1.6 Instantiating Black Box IP Cores with Generated VHDL Files

Use the syn_black_box compiler directive to declare a component as a black box. The top-level design files must contain the IP core variation component declaration and port-mapping. Apply the syn_black_box directive to the component declaration in the top-level file. The software compiles successfully without this directive, but reports an additional warning message. Using this directive allows you to add other directives.

The example shows a top-level file that instantiates **my_vhdlIP.vhd**, which is a simplified customized variation generated by the IP Catalog.

Example 124. Sample Top-Level VHDL Code with Black Box Instantiation of IP

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

ENTITY top IS
    PORT (
        clk: IN STD_LOGIC ;
        count: OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
    );
END top;

ARCHITECTURE rtl OF top IS
COMPONENT my_vhdlIP
    PORT (
        clock: IN STD_LOGIC ;
        q: OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
    );
end COMPONENT;
attribute syn_black_box : boolean;
attribute syn_black_box of my_vhdlIP: component is true;
BEGIN
    vhdlIP_inst : my_vhdlIP PORT MAP (
        clock => clk,
        q => count
    );
END rtl;
```

19.10.1.7 Other Synplify Software Attributes for Creating Black Boxes

Instantiating IP as a black box does not provide visibility into the IP for the synthesis tool. Thus, it does not take full advantage of the synthesis tool's timing-driven optimization. For better timing optimization, especially if the black box does not have registered inputs and outputs, add timing models to black boxes by adding the syn_tpd, syn_tsu, and syn_tco attributes.

Example 125. Adding Timing Models to Black Boxes in Verilog HDL

```
module ram32x4(z,d,addr,we,clk);
    /* synthesis syn_black_box syn_tcol="clk->z[3:0]=4.0"
        syn_tpd1="addr[3:0]->[3:0]=8.0"
        syn_tsu1="addr[3:0]->clk=2.0"
        syn_tsu2="we->clk=3.0" */
    output [3:0]z;
    input[3:0]d;
    input[3:0]addr;
```

```
    input we
    input clk
endmodule
```

The following additional attributes are supported by the Synplify software to communicate details about the characteristics of the black box module within the HDL code:

- `syn_resources`—Specifies the resources used in a particular black box.
- `black_box_pad_pin`—Prevents mapping to I/O cells.
- `black_box_tri_pin`—Indicates a tri-stated signal.

For more information about applying these attributes, refer to the *Synopsys FPGA Synthesis Reference Manual*.

19.10.2 Including Files for Quartus Prime Placement and Routing Only

In the Synplify software, you can add files to your project that are used only during placement and routing in the Quartus Prime software. This can be useful if you have grey or black boxes for Synplify synthesis that require the full design files to be compiled in the Quartus Prime software.

You can also set the option in a script using the `-job_owner par` option.

The example shows how to define files for a Synplify project that includes a top-level design file, a grey box netlist file, an IP wrapper file, and an encrypted IP file. With these files, the Synplify software writes an empty instantiation of "core" in the **.vqm** file and uses the grey box netlist for resource and timing estimation. The files **core.v** and **core_enc8b10b.v** are not compiled by the Synplify software, but are copied into the place-and-route directory. The Quartus Prime software compiles these files to implement the "core" IP block.

Example 126. Commands to Define Files for a Synplify Project

```
add_file -verilog -job_owner par "core_enc8b10b.v"
add_file -verilog -job_owner par "core.v"
add_file -verilog "core_gb.v"
add_file -verilog "top.v"
```

19.10.3 Inferring Intel FPGA IP Cores from HDL Code

The Synplify software uses Behavior Extraction Synthesis Technology (BEST) algorithms to infer high-level structures such as RAMs, ROMs, operators, FSMs, and DSP multiplication operations. Then, the Synplify software keeps the structures abstract for as long as possible in the synthesis process. This allows the use of technology-specific resources to implement these structures by inferring the appropriate Intel FPGA IP core when an IP core provides optimal results.

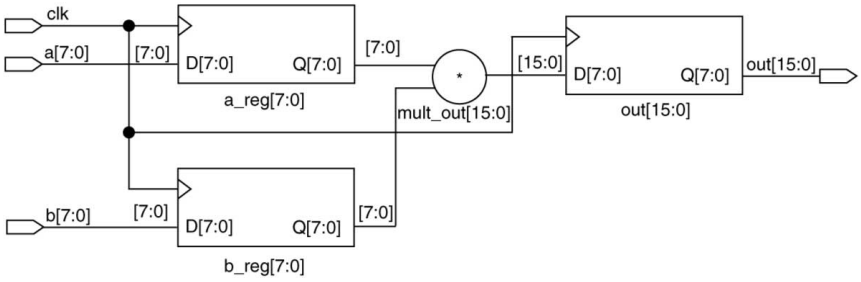
Related Links

Recommended HDL Coding Styles Documentation on page 102
This chapter provides Hardware Description Language (HDL) coding style recommendations to ensure optimal synthesis results when targeting Intel FPGA devices.

19.10.3.1 Inferring Multipliers

The figure shows the HDL Analyst view of an unsigned 8 × 8 multiplier with two pipeline stages after synthesis in the Synplify software. This multiplier is converted into an ALTMULT_ADD or ALTMULT_ACCUM IP core. For devices with DSP blocks, the software might implement the function in a DSP block instead of regular logic, depending on device utilization. For some devices, the software maps directly to DSP block device primitives instead of instantiating an IP core in the **.vqm** file.

Figure 303. HDL Analyst View of LPM_MULT IP Core (Unsigned 8x8 Multiplier with Pipeline=2)



19.10.3.1.1 Resource Balancing

While mapping multipliers to DSP blocks, the Synplify software performs resource balancing for optimum performance.

Intel devices have a fixed number of DSP blocks, which includes a fixed number of embedded multipliers. If the design uses more multipliers than are available, the Synplify software automatically maps the extra multipliers to logic elements (LEs), or adaptive logic modules (ALMs).

If a design uses more multipliers than are available in the DSP blocks, the Synplify software maps the multipliers in the critical paths to DSP blocks. Next, any wide multipliers, which might or might not be in the critical paths, are mapped to DSP blocks. Smaller multipliers and multipliers that are not in the critical paths might then be implemented in the logic (LEs or ALMs). This ensures that the design fits successfully in the device.

19.10.3.1.2 Controlling the DSP Block Inference

You can implement multipliers in DSP blocks or in logic in Intel devices that contain DSP blocks. You can control this implementation through attribute settings in the Synplify software.

19.10.3.1.3 Signal Level Attribute

You can control the implementation of individual multipliers by using the `syn_multstyle` attribute as shown in the following Verilog HDL code (where *<signal_name>* is the name of the signal):

```
<signal_name> /* synthesis syn_multstyle = "logic" */;
```

The `syn_multstyle` attribute applies to wires only; it cannot be applied to registers.

Table 269. DSP Block Attribute Setting in the Synplify Software

Attribute Name	Value	Description
syn_multstyle	lpm_mult	LPM function inferred and multipliers implemented in DSP blocks.
	logic	LPM function not inferred and multipliers implemented as LEs by the Synplify software.
	block_mult	DSP IP core is inferred and multipliers are mapped directly to DSP block device primitives (for supported devices).

Example 127. Signal Attributes for Controlling DSP Block Inference in Verilog HDL Code

```
module mult(a,b,c,r,en);
    input [7:0] a,b;
    output [15:0] r;
    input [15:0] c;
    input en;
    wire [15:0] temp /* synthesis syn_multstyle="logic" */;

    assign temp = a*b;
    assign r = en ? temp : c;
endmodule
```

Example 128. Signal Attributes for Controlling DSP Block Inference in VHDL Code

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity onereg is port (
    r : out std_logic_vector (15 downto 0);
    en : in std_logic;
    a : in std_logic_vector (7 downto 0);
    b : in std_logic_vector (7 downto 0);
    c : in std_logic_vector (15 downto 0);
    );
end onereg;

architecture beh of onereg is
signal temp : std_logic_vector (15 downto 0);
attribute syn_multstyle : string;
attribute syn_multstyle of temp : signal is "logic";

begin
    temp <= a * b;
    r <= temp when en='1' else c;
end beh;
```

19.10.3.2 Inferring RAM

When a RAM block is inferred from an HDL design, the Synplify software uses an Intel FPGA IP core to target the device memory architecture. For some devices, the Synplify software maps directly to memory block device primitives instead of instantiating an IP core in the `.vqm` file.

Follow these guidelines for the Synplify software to successfully infer RAM in a design:

- The address line must be at least two bits wide.
- Resets on the memory are not supported. Refer to the device family documentation for information about whether read and write ports must be synchronous.
- Some Verilog HDL statements with blocking assignments might not be mapped to RAM blocks, so avoid blocking statements when modeling RAMs in Verilog HDL.

For some device families, the `syn_ramstyle` attribute specifies the implementation to use for an inferred RAM. You can apply the `syn_ramstyle` attribute globally to a module or a RAM instance, to specify `registers` or `block_ram` values. To turn off RAM inference, set the attribute value to `registers`.

When inferring RAM for some Intel device families, the Synplify software generates additional bypass logic. This logic is generated to resolve a half-cycle read/write behavior difference between the RTL and post-synthesis simulations. The RTL simulation shows the memory being updated on the positive edge of the clock; the post-synthesis simulation shows the memory being updated on the negative edge of the clock. To eliminate bypass logic, the output of the RAM must be registered. By adding this register, the output of the RAM is seen after a full clock cycle, by which time the update has occurred, thus eliminating the need for bypass logic.

For devices with TriMatrix memory blocks, disable the creation of glue logic by setting the `syn_ramstyle` value to `no_rw_check`. Set `syn_ramstyle` to `no_rw_check` to disable the creation of glue logic in dual-port mode.

Example 129. VHDL Code for Inferred Dual-Port RAM

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

ENTITY dualport_ram IS
PORT ( data_out: OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
    data_in: IN STD_LOGIC_VECTOR (7 DOWNTO 0)
    wr_addr, rd_addr: IN STD_LOGIC_VECTOR (6 DOWNTO 0);
    we: IN STD_LOGIC);
    clk: IN STD_LOGIC);
END dualport_ram;

ARCHITECTURE ram_infer OF dualport_ram IS
TYPE Mem_Type IS ARRAY (127 DOWNTO 0) OF STD_LOGIC_VECOR (7 DOWNTO 0);
SIGNAL mem; Mem_Type;
SIGNAL addr_reg: STD_LOGIC_VECTOR (6 DOWNTO 0);

BEGIN
    data_out <= mem (CONV_INTEGER(rd_addr));
    PROCESS (clk, we, data_in) BEGIN
        IF (clk='1' AND clk'EVENT) THEN
            IF (we='1') THEN
                mem(CONV_INTEGER(wr_addr)) <= data_in;
            END IF;
        END IF;
    END PROCESS;
END ram_infer;
```

Example 130. VHDL Code for Inferred Dual-Port RAM Preventing Bypass Logic

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

ENTITY dualport_ram IS
PORT ( data_out: OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
    data_in : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
    wr_addr, rd_addr : IN STD_LOGIC_VECTOR (6 DOWNTO 0);
    we : IN STD_LOGIC;
    clk : IN STD_LOGIC);
END dualport_ram;

ARCHITECTURE ram_infer OF dualport_ram IS
TYPE Mem_Type IS ARRAY (127 DOWNTO 0) OF STD_LOGIC_VECTOR (7 DOWNTO 0);
SIGNAL mem : Mem_Type;
SIGNAL addr_reg : STD_LOGIC_VECTOR (6 DOWNTO 0);
SIGNAL tmp_out : STD_LOGIC_VECTOR (7 DOWNTO 0); --output register

BEGIN
    tmp_out <= mem (CONV_INTEGER (rd_addr));
    PROCESS (clk, we, data_in) BEGIN
        IF (clk='1' AND clk'EVENT) THEN
            IF (we='1') THEN
                mem(CONV_INTEGER(wr_addr)) <= data_in;
            END IF;
            data_out <= tmp_out; --registers output preventing
                                 -- bypass logic generation
        END IF;
    END PROCESS;
END ram_infer;
```

19.10.3.3 RAM Initialization

Use the Verilog HDL `$readmemb` or `$readmemh` system tasks in your HDL code to initialize RAM memories. The Synplify compiler forward-annotates the initialization values in the **.srs** (technology-independent RTL netlist) file and the mapper generates the corresponding hexadecimal memory initialization (**.hex**) file. One **.hex** file is created for each of the `altsyncram` IP cores that are inferred in the design. The **.hex** file is associated with the `altsyncram` instance in the **.vqm** file using the `init_file` attribute.

The examples show how RAM can be initialized through HDL code, and how the corresponding **.hex** file is generated using Verilog HDL.

Example 131. Using $readmemb System Task to Initialize an Inferred RAM in Verilog HDL Code

```
initial
begin
    $readmemb("mem.ini", mem);
end
always @(posedge clk)
begin
    raddr_reg <= raddr;
    if(we)
        mem[waddr] <= data;
end
```

Example 132. Sample of .vqm Instance Containing Memory Initialization File

```
altsyncram mem_hex( .wren_a(we),.wren_b(GND),...);

defparam mem_hex.lpm_type = "altsyncram";
defparam mem_hex.operation_mode = "Dual_Port";
...
defparam mem_hex.init_file = "mem_hex.hex";
```

19.10.3.4 Inferring ROM

When a ROM block is inferred from an HDL design, the Synplify software uses an Intel FPGA IP core to target the device memory architecture. For some devices, the Synplify software maps directly to memory block device atoms instead of instantiating an IP core in the **.vqm** file.

Follow these guidelines for the Synplify software to successfully infer ROM in a design:

- The address line must be at least two bits wide.
- The ROM must be at least half full.
- A CASE or IF statement must make 16 or more assignments using constant values of the same width.

19.10.3.5 Inferring Shift Registers

The Synplify software infers shift registers for sequential shift components so that they can be placed in dedicated memory blocks in supported device architectures using the ALTSHIFT_TAPS IP core.

If necessary, set the implementation style with the `syn_srlstyle` attribute. If you do not want the components automatically mapped to shift registers, set the value to `registers`. You can set the value globally, or on individual modules or registers.

For some designs, turning off shift register inference improves the design performance.

19.11 Document Revision History

Table 270. Document Revision History

Date	Version	Changes
2016.10.31	16.1.0	• Implemented Intel rebranding.
2016.05.03	16.0.0	• Removed support for NativeLink synthesis in Pro Edition
2015.11.02	15.1.0	• Changed instances of *Quartus II* to *Quartus Prime*.
2014.12.15	14.1.0	Updated location of Fitter Settings, Analysis & Synthesis Settings, and Physical Optimization Settings to Compiler Settings.
November 2013	13.1.0	Dita conversion. Restructured content.
June 2012	12.0.0	Removed survey link.
November 2011	10.1.1	Template update.

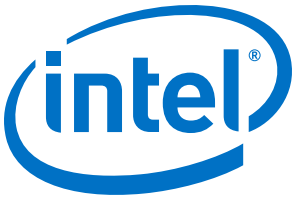
continued...

Date	Version	Changes
December 2010	10.1.0	• Changed to new document template. • Removed Classic Timing Analyzer support. • Removed the "altera_implement_in_esb or altera_implement_in_eab" section. • Edited the "Creating a Quartus Prime Project for Compile Points and Multiple .vqm Files" on page 14–33 section for changes with the incremental compilation flow. • Edited the "Creating a Quartus Prime Project for Multiple .vqm Files" on page 14–39 section for changes with the incremental compilation flow. • Editorial changes.
July 2010	10.0.0	• Minor updates for the Quartus Prime software version 10.0 release.
November 2009	9.1.0	• Minor updates for the Quartus Prime software version 9.1 release.
March 2009	9.0.0	• Added new section "Exporting Designs to the Quartus Prime Software Using NativeLink Integration" on page 14–14. • Minor updates for the Quartus Prime software version 9.0 release. • Chapter 10 was previously Chapter 9 in software version 8.1.
November 2008	8.1.0	• Changed to 8-1/2 x 11 page size • Changed the chapter title from "Synplicity Synplify & Synplify Pro Support" to "Synopsys Synplify Support" • Replaced references to Synplicity with references to Synopsys • Added information about Synplify Premier • Updated supported device list • Added SystemVerilog information to Figure 14–1
May 2008	8.0.0	• Updated supported device list • Updated constraint annotation information for the TimeQuest Timing Analyzer • Updated RAM and MAC constraint limitations • Revised Table 9–1 • Added new section "Changing Synplify's Default Behavior for Instantiated Altera Megafunctions" • Added new section "Instantiating Intellectual Property Using the MegaWizard Plug-In Manager and IP Toolbench" • Added new section "Including Files for Quartus Prime Placement and Routing Only" • Added new section "Additional Considerations for Compile Points" • Removed section "Apply the LogicLock Attributes" • Modified Figure 9–4, 9–43, 9–47. and 9–48 • Added new section "Performing Incremental Compilation in the Quartus Prime Software" • Numerous text changes and additions throughout the chapter • Renamed several sections • Updated "Referenced Documents" section

Related Links

Altera Documentation Archive

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



Intel® Quartus® Prime Pro Edition Handbook Volume 2: Design Implementation and Optimization

QPP5V2
2017.05.08


Subscribe


Send Feedback

Contents

1 Constraining Designs

Constraints, sometimes known as assignments or logic options, control the way the Quartus® Prime software implements a design for an FPGA. Constraints are also central in the way that the TimeQuest Timing Analyzer and the Power Analyzer inform synthesis, placement, and routing.

A Quartus Prime design contains several types of constraints:

- Global design constraints and software settings, such as device family selection, package type, and pin count.
- Entity-level constraints, such as logic options and placement assignments.
- Instance-level constraints.
- Pin assignments and I/O constraints.

Quartus Prime software stores user-created constraints in one of two files: the Quartus Prime Settings File (`.qsf`) or, in the case of timing constraints, the Synopsys* Design Constraints file (`.sdc`).

In the Quartus Prime software you can constrain designs for compilation and analysis using the GUI or using Tcl syntax and scripting. By combining the Tcl syntax of the `.qsf` files and the `.sdc` files with procedural Tcl, you can automate iteration over several different settings, changing constraints and recompiling.

Quartus Prime Settings Files

Quartus Prime settings files contain project-wide and instance-level assignments for the current revision of the project, in Tcl syntax. Each revision of your project has a separate `.qsf` file.

Quartus Prime software stores constraints and assignments you make with the **Device** dialog box, **Settings** dialog box, Assignment Editor, Chip Planner, and Pin Planner in the Quartus Prime Settings File.

Synopsys Design Constraints Files

The TimeQuest Timing Analyzer uses industry-standard Synopsys Design Constraints, and stores those constraints in Synopsys Design Constraints (`.sdc`) files.

1.1 Constraining Designs with the GUI

In the Quartus Prime GUI, the New Project Wizard, **Device** dialog box, and **Settings** dialog box allow you to make global constraints and software settings. The Assignment Editor and Pin Planner are spreadsheet-style interfaces for constraining your design at the instance or entity level.

The Assignment Editor and Pin Planner make constraint types and values available based on global design characteristics such as the targeted device. These tools help you verify that your constraints are valid before compilation by allowing you to pick only from valid values for each constraint.

The TimeQuest Timing Analyzer GUI allows you to make timing constraints in SDC format and view the effects of those constraints on the timing in your design. Before running the TimeQuest timing analyzer, you must specify initial timing constraints that describe the clock characteristics, timing exceptions, and external signal arrival and required times. The Quartus Prime Fitter optimizes the placement of logic in the device to meet your specified constraints.

1.1.1 Global Constraints

Global constraints affect the entire Quartus Prime project and all the applicable logic in the design. Many of these constraints are simply project settings, such as the target device you select for the design.

You can also globally apply synthesis optimizations and timing and power analysis settings. You often define global constraints in early project development; for example, when running the New Project Wizard.

To specify global constraints, use the **Device** or the **Settings** dialog boxes.

1.1.1.1 Common Types of Global Constraints

The following are the most common types of global constraints:

- Target device specification
- Top-level entity of your design, and the names of the design files included in the project
- Operating temperature limits and conditions
- Physical synthesis optimizations
- Analysis and synthesis options and optimization techniques
- Verilog HDL and VHDL language versions used in your project
- Timing driven compilation settings
- `.sdc` files for the TimeQuest timing analyzer to use during analysis as part of a full compilation flow

1.1.1.2 Settings That Direct Compilation and Analysis Flows

Settings that direct compilation and analysis flows in the Quartus Prime software are stored in the Quartus Prime Settings File for your project, including the following global software settings:

- Settings for EDA tool integration such as third-party synthesis tools, simulation tools, timing analysis tools, and formal verification tools.
- Settings and settings file specifications for the Quartus Prime Assembler, Signal Tap Logic Analyzer, and power analyzer.

1.1.1.3 Global Constraints and Software Settings

Global constraints and software settings stored in the Quartus Prime settings file are specific to each revision of your design, allowing you to control the operation of the software differently for different revisions. For example, different revisions can specify different operating temperatures and different devices, so that you can compare results.

Only the valid assignments made in the Assignment Editor are saved in the Quartus Prime Settings File, which is located in the project directory. When you make a design constraint, the new assignment is placed on a new line at the end of the file.

When you create or update a constraint in the GUI, the Quartus Prime software displays the equivalent Tcl command in the **System** tab of the **Messages** window. You can use the displayed messages as references when making assignments using Tcl commands.

Related Links

Quartus Prime Pro Edition Settings File Reference Manual
For information about all settings and constraints in the Quartus Prime software.

1.1.2 Node, Entity, and Instance-Level Constraints

Node, entity, and instance-level constraints apply to a particular segment of the design hierarchy, as opposed to the entire design. In the Quartus Prime software, you specify most instance-level constraints with the Assignment Editor, Pin Planner, and Chip Planner.

The Assignment Editor and the Pin Planner aid you in correctly constraining your design through device-and-assignment-determined pick lists and through live I/O checking.

You can assign logic functions to physical resources on the device using location assignments with the Assignment Editor or the Chip Planner. Node, entity, and instance-level constraints take precedence over any global constraints that affect the same sections of the design hierarchy. You can edit and view all node and entity-level constraints you created in the Assignment Editor, or you can filter the assignments by filtering assignments only for specific locations, such as DSP blocks.

1.1.2.1 Constraining Designs with the Pin Planner

The Pin Planner helps you visualize, plan, and assign device I/O pins to ensure compatibility with your PCB layout. The Pin Planner provides a graphical view of the I/O resources in the target device package. You can quickly locate various I/O pins and assign them design elements or other properties.

The Quartus Prime software uses these assignments to place and route your design during device programming. The Pin Planner also helps with early pin planning by allowing you to plan and assign IP interface or user nodes not yet defined in the design.

The Pin Planner **Task** window provides one-click access to common pin planning tasks. After clicking a pin planning task, you view and highlight the results in the **Report** window by selecting or deselecting I/O types. You can quickly identify I/O banks, VREF groups, edges, and differential pin pairings to assist you in the pin planning process.

1.1.2.2 Constraining Designs with the Chip Planner

The Chip Planner allows you to view the device from a variety of different perspectives, and make precise assignments to specific floorplan locations.

With the Chip Planner, you can adjust existing assignments to device resources, such as pins, logic cells, and LABs using drag and drop features and a graphical interface. You can also view equations and routing information, and demote assignments by dragging and dropping assignments to various regions in the **Regions** window.

Related Links

Design Floorplan Analysis in the Chip Planner on page 190
The Chip Planner simplifies floorplan analysis by providing visual display of chip resources.

1.1.3 Probing Between Components of the Quartus Prime GUI

The Assignment Editor, Chip Planner, and Pin Planner let you locate nodes and instances in the source files for your design in other Quartus Prime viewers.

You can select a cell in the Assignment Editor spreadsheet and locate the corresponding item in another applicable Quartus Prime software window, such as the Chip Planner. To locate an item from the Assignment Editor in another window, right-click the item of interest in the spreadsheet, point to **Locate**, and click the appropriate command.

You can also locate nodes in the Assignment Editor and other constraint tools from other windows within the Quartus Prime software. First, select the node or nodes in the appropriate window. For example, select an entity in the **Entity** list in the **Hierarchy** tab in the Project Navigator, or select nodes in the Chip Planner. Next, right-click the selected object, point to **Locate**, and click **Locate in Assignment Editor**. The Assignment Editor opens, or it is brought to the foreground if it is already open.

1.1.4 SDC and the TimeQuest Timing Analyzer

You can make individual timing constraints for individual entities, nodes, and pins with the **Constraints** menu of the TimeQuest Timing Analyzer. The TimeQuest Timing Analyzer GUI provides easy access to timing constraints, and reporting, without requiring knowledge of SDC syntax.

As you specify commands and options in the GUI, the corresponding SDC or Tcl command appears in the Console. This lets you know exactly what constraint you added to your Synopsys Design Constraints file, and also enables you to learn SDC syntax for use in scripted flows. The GUI also provides enhanced graphical reporting features.

Individual timing assignments override project-wide requirements. You can also assign timing exceptions to nodes and paths to avoid reporting of incorrect or irrelevant timing violations. The TimeQuest timing analyzer supports point-to-point timing constraints, wildcards to identify specific nodes when making constraints, and assignment groups to make individual constraints to groups of nodes.

1.2 Constraining Designs with Tcl

Because `.sdc` files and `.qsf` files are both in Tcl syntax, you can modify these files to be part of a scripted constraint and compilation flow.

With Quartus Prime Tcl packages, Tcl scripts can open projects, make the assignments procedurally that would otherwise be specified in a `.qsf` file, compile a design, and compare compilation results against known goals and benchmarks for the design. Such a script can further automate the iterative process by modifying design constraints and recompiling the design.

1.2.1 Quartus Prime Settings Files and Tcl

QSF files use Tcl syntax, but, unmodified, are not executable scripts. However, you can embed QSF constraints in a scripted iterative compilation flow, where the script that automates compilation and custom results reporting also contains the design constraints.

```
set_global_assignment -name FAMILY "Cyclone II"
set_global_assignment -name DEVICE EP2C35F672C6
set_global_assignment -name TOP_LEVEL_ENTITY chiptrip
set_global_assignment -name ORIGINAL_QUARTUS_VERSION 10.0
set_global_assignment -name PROJECT_CREATION_TIME_DATE "11:45:02  JUNE 08, 2010"
set_global_assignment -name LAST_QUARTUS_VERSION 10.0
set_global_assignment -name MIN_CORE_JUNCTION_TEMP 0
set_global_assignment -name MAX_CORE_JUNCTION_TEMP 85
set_instance_assignment -name PARTITION_HIERARCHY root_partition -to | -
section_id Top
set_global_assignment -name PARTITION_NETLIST_TYPE SOURCE -section_id Top
set_global_assignment -name PARTITION_FITTER_PRESERVATION_LEVEL
PLACEMENT_AND_ROUTING \ -section_id Top
set_global_assignment -name PARTITION_COLOR 16764057 -section_id Top
set_global_assignment -name LL_ROOT_REGION ON -section_id "Root Region"
set_global_assignment -name LL_MEMBER_STATE LOCKED -section_id "Root Region"
set_global_assignment -name STRATIX_DEVICE_IO_STANDARD "3.3-V LVTTL"
set_location_assignment PIN_P2 -to clk2
set_location_assignment PIN_AE4 -to ticket[0]
set_location_assignment PIN_J23 -to ticket[2]
set_location_assignment PIN_Y12 -to timeo[1]
set_location_assignment PIN_N2 -to reset
set_location_assignment PIN_R2 -to timeo[7]
set_location_assignment PIN_P1 -to clk1
set_location_assignment PIN_M3 -to ticket[1]
set_location_assignment PIN_AE24 -to ~LVDS150p/nCEO~
set_location_assignment PIN_C2 -to accel
set_location_assignment PIN_K4 -to ticket[3]
set_location_assignment PIN_B3 -to stf
set_location_assignment PIN_T9 -to timeo[0]
set_location_assignment PIN_M5 -to timeo[6]
set_location_assignment PIN_J8 -to dir[1]
set_location_assignment PIN_C5 -to timeo[5]
set_location_assignment PIN_F6 -to gt1
set_location_assignment PIN_P24 -to timeo[2]
set_location_assignment PIN_B2 -to at_altera
set_location_assignment PIN_P3 -to timeo[4]
set_location_assignment PIN_M4 -to enable
set_location_assignment PIN_E3 -to ~ASDO~
set_location_assignment PIN_E5 -to dir[0]
set_location_assignment PIN_R25 -to timeo[3]
set_location_assignment PIN_D3 -to ~nCSO~
set_location_assignment PIN_G4 -to gt2
set_global_assignment -name MISC_FILE "D:/altera/chiptrip/chiptrip.dpf"
set_global_assignment -name USE_TIMEQUEST_TIMING_ANALYZER ON
set_global_assignment -name POWER_PRESET_COOLING_SOLUTION \
```

```
"23 MM HEAT SINK WITH 200 LFPM AIRFLOW"
set_global_assignment -name POWER_BOARD_THERMAL_MODEL "NONE (CONSERVATIVE)"
set_global_assignment -name SDC_FILE chiptrip.sdc
```

The example shows the way that the `set_global_assignment` Quartus Prime Tcl command makes all global constraints and software settings, with `set_location_assignment` constraining each I/O node in the design to a physical pin on the device.

However, after you initially create the Quartus Prime Settings File for your design, you can export the contents to a procedural, executable Tcl (`.tcl`) file. You can then use that generated script to restore certain settings after experimenting with other constraints. You can also use the generated Tcl script to archive your assignments instead of archiving the Quartus Prime Settings file itself.

To export your constraints as an executable Tcl script, click **Project ➤ Generate Tcl File for Project**.

```
# Quartus Prime: Generate Tcl File for Project
# File: chiptrip.tcl
# Generated on: Tue Jun 08 13:08:48 2010
# Load Quartus Prime Tcl Project package
package require ::quartus::project
set need_to_close_project 0
set make_assignments 1
# Check that the right project is open
if {[is_project_open]} {
    if {[string compare $quartus(project) "chiptrip"]} {
        puts "Project chiptrip is not open"
        set make_assignments 0
    }
} else {
    # Only open if not already open
    if {[project_exists chiptrip]} {
        project_open -revision chiptrip chiptrip
    } else {
        project_new -revision chiptrip chiptrip
    }
    set need_to_close_project 1
}
# Make assignments
if {$make_assignments} {
set_global_assignment -name FAMILY "Cyclone®  II"
set_global_assignment -name DEVICE EP2C35F672C6
set_global_assignment -name TOP_LEVEL_ENTITY chiptrip
set_global_assignment -name ORIGINAL_QUARTUS_VERSION 10.0
set_global_assignment -name PROJECT_CREATION_TIME_DATE "11:45:02  JUNE 08, 2010"
set_global_assignment -name LAST_QUARTUS_VERSION 10.0
set_global_assignment -name MIN_CORE_JUNCTION_TEMP 0
set_global_assignment -name MAX_CORE_JUNCTION_TEMP 85
set_instance_assignment -name PARTITION_HIERARCHY root_partition -to | -
section_id Top
set_global_assignment -name PARTITION_NETLIST_TYPE SOURCE -section_id Top
set_global_assignment -name PARTITION_FITTER_PRESERVATION_LEVEL
PLACEMENT_AND_ROUTING \ -section_id Top
set_global_assignment -name PARTITION_COLOR 16764057 -section_id Top
set_global_assignment -name LL_ROOT_REGION ON -section_id "Root Region"
set_global_assignment -name LL_MEMBER_STATE LOCKED -section_id "Root Region"
set_global_assignment -name STRATIX_DEVICE_IO_STANDARD "3.3-V LVTTL"
set_location_assignment PIN_P2 -to clk2
set_location_assignment PIN_AE4 -to ticket[0]
set_location_assignment PIN_J23 -to ticket[2]
set_location_assignment PIN_Y12 -to timeo[1]
set_location_assignment PIN_N2 -to reset
set_location_assignment PIN_R2 -to timeo[7]
set_location_assignment PIN_P1 -to clk1
```

```
set_location_assignment PIN_M3 -to ticket[1]
set_location_assignment PIN_AE24 -to ~LVDS150p/nCEO~
set_location_assignment PIN_C2 -to accel
set_location_assignment PIN_K4 -to ticket[3]
set_location_assignment PIN_B3 -to stf
set_location_assignment PIN_T9 -to timeo[0]
set_location_assignment PIN_M5 -to timeo[6]
set_location_assignment PIN_J8 -to dir[1]
set_location_assignment PIN_C5 -to timeo[5]
set_location_assignment PIN_F6 -to gt1
set_location_assignment PIN_P24 -to timeo[2]
set_location_assignment PIN_B2 -to at_altera
set_location_assignment PIN_P3 -to timeo[4]
set_location_assignment PIN_M4 -to enable
set_location_assignment PIN_E3 -to ~ASDO~
set_location_assignment PIN_E5 -to dir[0]
set_location_assignment PIN_R25 -to timeo[3]
set_location_assignment PIN_D3 -to ~nCSO~
set_location_assignment PIN_G4 -to gt2
set_global_assignment -name MISC_FILE "D:/altera/chiptrip/chiptrip.dpf"
set_global_assignment -name USE_TIMEQUEST_TIMING_ANALYZER ON
set_global_assignment -name POWER_PRESET_COOLING_SOLUTION \
"23 MM HEAT SINK WITH 200 LFPM AIRFLOW"
set_global_assignment -name POWER_BOARD_THERMAL_MODEL "NONE (CONSERVATIVE)"
set_global_assignment -name SDC_FILE chiptrip.sdc
    # Commit assignments
    export_assignments
    # Close project
    if {$need_to_close_project} {
        project_close
    }
}
```

After setting initial values for variables to control constraint creation and whether or not the project needs to be closed at the end of the script, the generated script checks to see if a project is open. If a project is open but it is not the correct project, in this case, **chiptrip**, the script prints `Project chiptrip is not open` to the console and does nothing else.

If no project is open, the script determines if **chiptrip** exists in the current directory. If the project exists, the script opens the project. If the project does not exist, the script creates a new project and opens the project.

The script then creates the constraints. After creating the constraints, the script writes the constraints to the Quartus Prime Settings File and then closes the project.

1.2.2 Timing Analysis with Synopsys Design Constraints and Tcl

Quartus Prime software uses `.sdc` files to save the timing constraints TimeQuest Timing Analyzer uses. Since `.sdc` files use Tcl syntax, you can incorporate the constraints into other scripts for iterative timing analysis.

```
# ------------------------------------------
set_time_unit ns
set_decimal_places 3
# ------------------------------------------
#
create_clock -period 10.0 -waveform { 0 5.0 } clk2 -name clk2
create_clock -period 4.0 -waveform { 0 2.0 } clk1 -name clk1
# clk1 -> dir* : INPUT_MAX_DELAY = 1 ns
set_input_delay -max 1ns -clock clk1  [get_ports dir*]
# clk2 -> time* : OUTPUT_MAX_DELAY = -2 ns
set_output_delay -max -2ns -clock clk2  [get_ports time*]
```

Similar to the constraints in the Quartus Prime Settings File, you can make the SDC constraints part of an executable timing analysis script.

The clock settings and delay constraints in the following script are identical to those in the `.sdc` file from the first example. The script:

1. Opens the project
2. Creates a timing netlist
3. Constrains the two clocks in the design
4. Applies input and output delay constraints
5. Updates the timing netlist for the constraints
6. Performs multi-corner timing analysis on the design

```
project_open chiptrip
create_timing_netlist
#
# Create Constraints
#
create_clock -period 10.0 -waveform { 0 5.0 } clk2 -name clk2
create_clock -period 4.0 -waveform { 0 2.0 } clk1 -name clk1
# clk1 -> dir* : INPUT_MAX_DELAY = 1 ns
set_input_delay -max 1ns -clock clk1  [get_ports dir*]
# clk2 -> time* : OUTPUT_MAX_DELAY = -2 ns
set_output_delay -max -2ns -clock clk2  [get_ports time*]
#
# Perform timing analysis for several different sets of operating conditions
#
foreach_in_collection oc [get_available_operating_conditions] {
    set_operating_conditions $oc
    update_timing_netlist
    report_timing -setup -npaths 1
    report_timing -hold -npaths 1
    report_timing -recovery -npaths 1
    report_timing -removal -npaths 1
    report_min_pulse_width -nworst 1
}
delete_timing_netlist
project_close
```

1.3 A Fully Iterative Scripted Flow

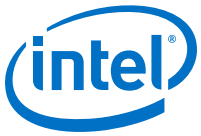
You can use the **::quartus::flow** Tcl package and other packages in the Quartus Prime Tcl API to add flow control to modify constraints and recompile your design in an automated flow. You can combine your timing constraints with other design constraints, and embed them in an executable Tcl script that iteratively compiles your design as you apply different constraints.

Each time such a modified generated script runs, it can modify your project's `.qsf` and `.sdc` files based on the iterative results.

This type of scripted flow can include automated design compilation, constraint modification, and recompilation based on how you foresee results and pre-determine next-step changes.

Related Links

- API Functions for Tcl
  In Quartus Prime Help



- Tcl Scripting on page 72

1.4 Document Revision History

Table 1. Document Revision History

Date	Version	Changes
2017.05.08	17.0.0	• Removed references to deprecated Fitter Effort logic option.
2016.10.31	16.1.0	• Implemented Intel rebranding.
2015.11.02	15.1.0	• Changed instances of *Quartus II* to *Quartus Prime*.
June 2014	14.0.0	Formatting updates.
November 2012	12.1.0	Update Pin Planner description for task and report windows.
June 2012	12.0.0	Removed survey link.
November 2011	10.0.2	Template update.
December 2010	10.0.1	Template update.
July 2010	10.0.0	Rewrote chapter to more broadly cover all design constraint methods. Removed procedural steps and user interface details, and replaced with links to Quartus Prime Help.
November 2009	9.1.0	• Added two notes. • Minor text edits.
March 2009	9.0.0	• Revised and reorganized the entire chapter. • Added section "Probing to Source Design Files and Other Quartus Prime Windows" on page1–2. • Added description of node type icons (Table1–3). • Added explanation of wildcard characters.
November 2008	8.1.0	Changed to 8½" × 11" page size. No change to content.
May 2008	8.0.0	Updated Quartus Prime software 8.0 revision and date.

Related Links

Altera Documentation Archive

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

2 Managing Device I/O Pins

This chapter describes efficient planning and assignment of I/O pins in your target device. Consider I/O standards, pin placement rules, and your PCB characteristics early in the design phase.

Figure 1. Pin Planner GUI

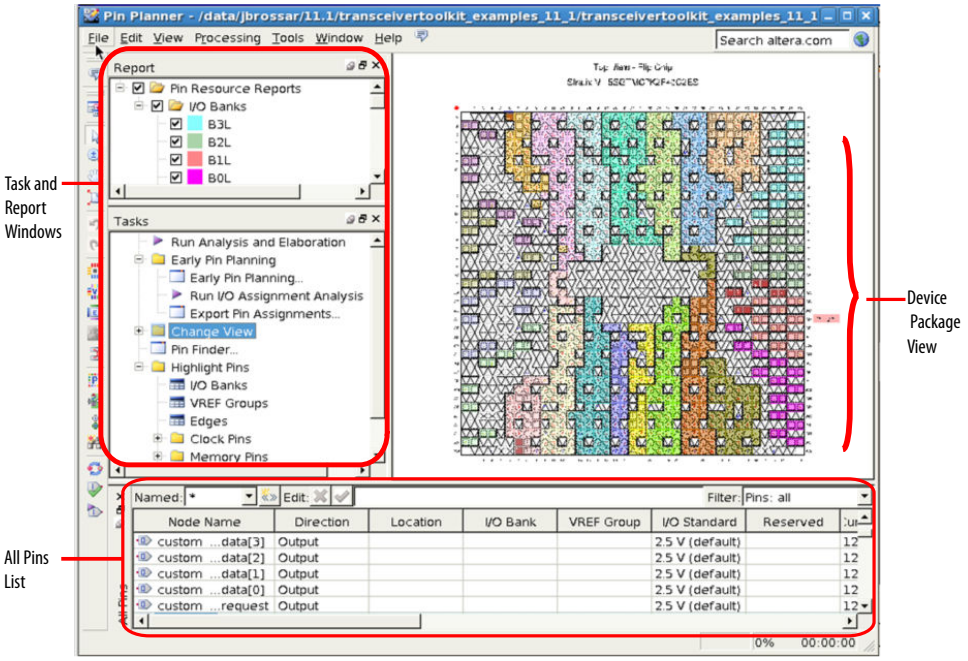


Table 2. Quartus Prime I/O Pin Planning Tools

I/O Planning Task	Click to Access
Plan interfaces and device periphery	**Tools ➤ BluePrint Platform Designer**
Edit, validate, or export pin assignments	**Assignments ➤ Pin Planner**

For more information about special pin assignment features for the Arria® 10 SoC devices, refer to *Instantiating the HPS Component* in the *Arria 10 Hard Processor System Technical Reference Manual*.

Related Links

- Instantiating the HPS Component
  In *Arria 10 Hard Processor System Technical Reference Manual*
- BluePrint Planning Overview on page 42
  After design synthesis, use BluePrint to rapidly define a legal device floorplan.

2.1 I/O Planning Overview

Plan and assign I/O pins in your design for compatibility with your target device and PCB characteristics. Plan I/O pins early to reduce design iterations and develop an accurate PCB layout sooner. You can assign expected nodes not yet defined in design files, including interface IP core signals, and then generate a top-level file. Specify interfaces for memory, high-speed I/O, device configuration, and debugging tools in your top-level file. The top-level file instantiates the next level of design hierarchy and includes interface port information.

Use the Pin Planner to view, assign, and validate device I/O pin logic and properties. Alternatively, you can enter I/O assignments in a Tcl script, or directly in HDL code. The Pin Planner **Task** window provides one-click access to I/O planning steps. You can filter and search the nodes in the design. You can define custom groups of pins for assignment. Instantly locate and highlight specific pin types for assignment or evaluation, such as I/O banks, VREF groups, edges, DQ/DQS pins, hard memory interface pins, PCIe hard IP interface pins, hard processor system pins, and clock region input pins. Assign design elements, I/O standards, interface IP, and other properties to the device I/O pins by name or by drag and drop. You can then generate a top-level design file for I/O validation.

Use I/O assignment analysis to fully analyze I/O analysis against VCCIO, VREF, electromigration (current density), Simultaneous Switching Output (SSO), drive strength, I/O standard, PCI_IO clamp diode, and I/O pin direction compatibility rules.

2.1.1 Basic I/O Planning Flow

The following steps describe the basic flow for assigning and verifying I/O pin assignments:

1. Click **Assignments ➤ Device** and select a target device that meets your logic, performance, and I/O requirements. Consider and specify I/O standards, voltage and power supply requirements, and available I/O pins.
2. Click **Assignments ➤ Pin Planner**.
3. Assign I/O properties to match your device and PCB characteristics, including assigning logic, I/O standards, output loading, slew rate, and current strength.
4. Click **Run I/O Assignment Analysis** in the **Tasks** pane to validate assignments and generate a synthesized design netlist. Correct any problems reported.
5. Click **Processing ➤ Start Compilation**. During compilation, the Quartus Prime software runs I/O assignment analysis.

2.1.2 Integrating PCB Design Tools

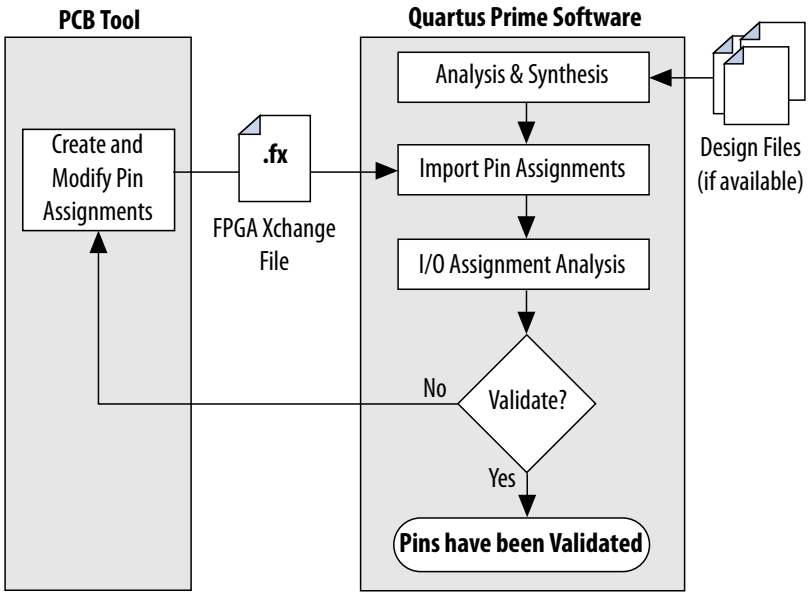
You can integrate PCB design tools into your work flow to map pin assignments to symbols in your system circuit schematics and board layout.

The Quartus Prime software integrates with board layout tools by allowing import and export of pin assignment information in Quartus Prime Settings Files (`.qsf`), Pin-Out File (`.pin`), and FPGA Xchange-Format File (`.fx`) files.

Table 3. Integrating PCB Design Tools

PCB Tool Integration	Supported PCB Tool
Define and validate I/O assignments in the Pin Planner, and then export the assignments to the PCB tool for validation	Mentor Graphics* I/O Designer Cadence Allegro
Define I/O assignments in your PCB tool, and then import the assignments into the Pin Planner for validation	Mentor Graphics I/O Designer Cadence Allegro

Figure 2. PCB Tool Integration

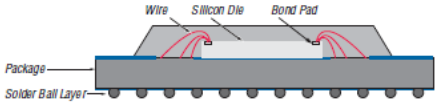
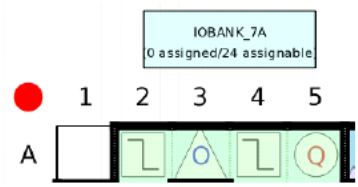
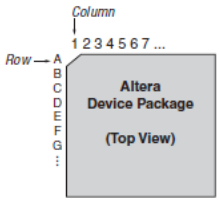
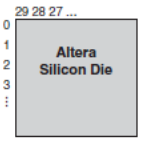
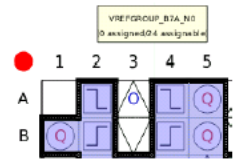


Related Links

- Mentor Graphics PCB Design Tools Support on page 279
  You can integrate the Mentor Graphics ® I/O Designer or DxDesigner PCB design tools into the Quartus Prime design flow. This combination provides a complete FPGA-to-board design workflow.
- Cadence PCB Design Tools Support on page 261

2.1.3 Intel Device Terms

The following terms describe Intel device and I/O structures:

Terms	Description	Diagram
Device Package (BGA example)	Ceramic or plastic heat sink surface mounted with FPGA die and I/O pins or solder balls. In a wire bond BGA example, copper wires connect the bond pads to the solder balls of the package. Click **View > Show > Package Top** or **View > Show > Package Bottom** in Pin Planner	
I/O Bank	I/O pins are grouped in I/O banks for assignment of I/O standards. Each numbered bank has its own voltage source pins, called VCCIO pins, for high I/O performance. The specified VCCIO pin voltage is between 1.5 V and 3.3 V. Each bank supports multiple pins with different I/O standards. All pins in a bank must use the same VCCIO signal. Click **View > Show > I/O Banks** in Pin Planner.	
I/O Pin	A wire lead or small solder ball on the package bottom or periphery. Each pin has an alphanumeric row and column number. I, O, Q, S, X, and Z are never used. The alphabet is repeated and prefixed with the letter A when exceeded. All I/O pins display by default.	
Pad	I/O pins are connected to pads located on the perimeter of the top metal layer of the silicon die. Each pad is numbered with an ID starting at 0, and increments by one in a counterclockwise direction around the device. Click **View > Pad View** in Pin Planner.	
VREF Pin Group	A group of pins including one dedicated VREF pin required by voltage-referenced I/O standards. A VREF group contains a smaller number of pins than an I/O bank. This maintains the signal integrity of the VREF pin. One or more VREF groups exist in an I/O bank. The pins in a VREF group share the same VCCIO and VREF voltages. Click **View > Show > Show VREF Groups** in Pin Planner..	

2.2 Assigning I/O Pins

Use the Pin Planner to visualize, modify, and validate I/O assignments in a graphical representation of the target device. You can increase the accuracy of I/O assignment analysis by reserving specific device pins to accommodate undefined but expected I/O.

To assign I/O pins in the Pin Planner, follow these steps:

1. Open a Quartus Prime project, and then click **Assignments ➤ Pin Planner**.
2. Click **Processing ➤ Start Analysis & Elaboration** to elaborate the design and display **All Pins** in the device view.
3. To locate or highlight pins for assignment, click **Pin Finder** or a pin type under **Highlight Pins** in the **Tasks** pane.
4. (Optional) To define a custom group of nodes for assignment, select one or more nodes in the **Groups** or **All Pins** list, and click **Create Group**.
5. Enter assignments of logic, I/O standards, interface IP, and properties for device I/O pins in the **All Pins** spreadsheet, or by drag and drop into the package view.
6. To assign properties to differential pin pairs, click **Show Differential Pin Pair Connections**. A red connection line appears between positive (p) and negative (n) differential pins.
7. (Optional) To create board trace model assignments:
   a. Right-click an output or bidirectional pin, and click **Board Trace Model**. For differential I/O standards, the board trace model uses a differential pin pair with two symmetrical board trace models.
   b. Specify board trace parameters on the positive end of the differential pin pair. The assignment applies to the corresponding value on the negative end of the differential pin pair.
8. To run a full I/O assignment analysis, click **Run I/O Assignment Analysis**. The Fitter reports analysis results. Only reserved pins are analyzed prior to design synthesis.

2.2.1 Assigning to Exclusive Pin Groups

You can designate groups of pins for exclusive assignment. When you assign pins to an **Exclusive I/O Group**, the Fitter does not place the signals in the same I/O bank with any other exclusive I/O group. For example, if you have a set of signals assigned exclusively to `group_a`, and another set of signals assigned to `group_b`, the Fitter ensures placement of each group in different I/O banks.

2.2.2 Assigning Slew Rate and Drive Strength

You can designate the device pin slew rate and drive strength. These properties affect the pin's outgoing signal integrity. Use either the **Slew Rate** or **Slow Slew Rate** assignment to adjust the drive strength of a pin with the **Current Strength** assignment.

Note: The slew rate and drive strength apply during I/O assignment analysis.

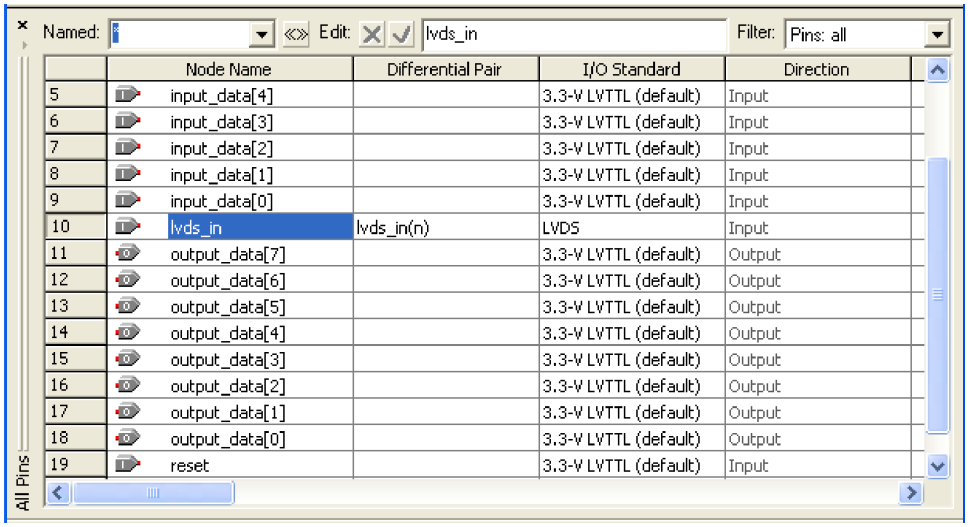
2.2.3 Assigning Differential Pins

When you assign a differential I/O standard to a single-ended top-level pin in your design, the Pin Planner automatically recognizes the negative pin as part of the differential pin pair assignment and creates the negative pin for you. The Quartus Prime software writes the location assignment for the negative pin to the `.qsf`; however, the I/O standard assignment is not added to the `.qsf` for the negative pin of the differential pair.

The following example shows a design with `lvds_in` top-level pin, to which you assign a differential I/O standard. The Pin Planner automatically creates the differential pin, `lvds_in(n)` to complete the differential pin pair.

Note: If you have a single-ended clock that feeds a PLL, assign the pin only to the positive clock pin of a differential pair in the target device. Single-ended pins that feed a PLL and are assigned to the negative clock pin device cause the design to not fit.

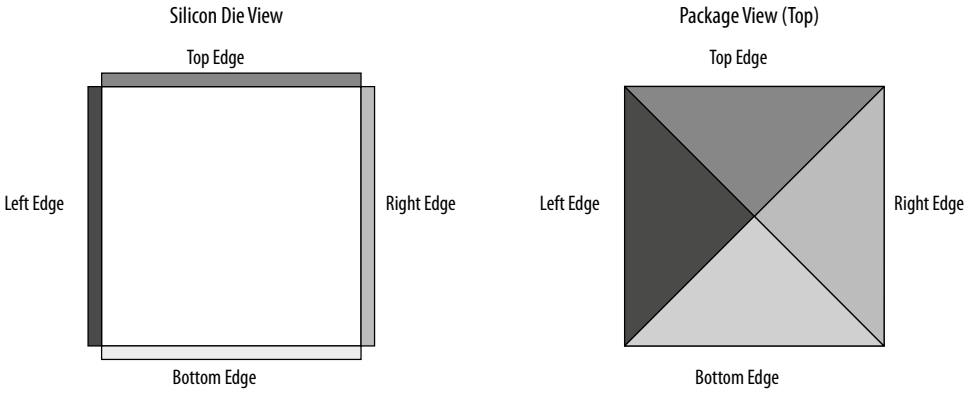
Figure 3. Creating a Differential Pin Pair in the Pin Planner



	Node Name	Differential Pair	I/O Standard	Direction
5	input_data[4]		3.3-V LVTTL (default)	Input
6	input_data[3]		3.3-V LVTTL (default)	Input
7	input_data[2]		3.3-V LVTTL (default)	Input
8	input_data[1]		3.3-V LVTTL (default)	Input
9	input_data[0]		3.3-V LVTTL (default)	Input
10	lvds_in	lvds_in(n)	LVDS	Input
11	output_data[7]		3.3-V LVTTL (default)	Output
12	output_data[6]		3.3-V LVTTL (default)	Output
13	output_data[5]		3.3-V LVTTL (default)	Output
14	output_data[4]		3.3-V LVTTL (default)	Output
15	output_data[3]		3.3-V LVTTL (default)	Output
16	output_data[2]		3.3-V LVTTL (default)	Output
17	output_data[1]		3.3-V LVTTL (default)	Output
18	output_data[0]		3.3-V LVTTL (default)	Output
19	reset		3.3-V LVTTL (default)	Input

If your design contains a large bus that exceeds the pins available in a particular I/O bank, you can use edge location assignments to place the bus. Edge location assignments improve the circuit board routing ability of large buses, because they are close together near an edge. The following figure shows Intel device package edges.

Figure 4. Die View and Package View of the Four Edges on an Intel Device



2.2.3.1 Overriding I/O Placement Rules on Differential Pins

I/O placement rules ensure that noisy signals do not corrupt neighboring signals. Each device family has predefined I/O placement rules.

I/O placement rules define, for example, the allowed placement of single-ended I/O with respect to differential pins, or how many output and bidirectional pins you can place within a VREF group when using voltage referenced input standards.

Use the IO_MAXIMUM_TOGGLE_RATE assignment to override I/O placement rules on pins, such as system reset pins that do not switch during normal design activity. Setting a value of 0 MHz for this assignment causes the Fitter to recognize the pin at a DC state throughout device operation. The Fitter excludes the assigned pin from placement rule analysis. Do not assign an IO_MAXIMUM_TOGGLE_RATE of 0 MHz to any actively switching pin, or your design may not function as you intend.

2.2.4 Entering Pin Assignments with Tcl Commands

You can use Tcl scripts to apply pin assignments rather than using the GUI. Enter individual Tcl commands in the Tcl Console, or type the following to apply the assignments contained in a Tcl script:

Example 1. Applying Tcl Script Assignments

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

Example 2. Scripted Pin Assignment

The following example uses `set_location_assignment` and `set_instance_assignment` Tcl commands to assign a pin to a specific location, I/O standard, and drive strength.

```
set_location_assignment PIN M20 -to address[10]
set_instance_assignment -name IO_STANDARD "2.5 V" -to address[10]
set_instance_assignment -name
        CURRENT_STRENGTH_NEW "MAXIMUM CURRENT" -to address[10]
```

Related Links

Tcl Scripting on page 72

2.2.5 Entering Pin Assignments in HDL Code

You can use synthesis attributes or low-level I/O primitives to embed I/O pin assignments directly in your HDL code. When you analyze and synthesize the HDL code, the information is converted into the appropriate I/O pin assignments. You can use either of the following methods to specify pin-related assignments with HDL code:

- Assigning synthesis attributes for signal names that are top-level pins
- Using low-level I/O primitives, such as ALT_BUF_IN, to specify input, output, and differential buffers, and for setting parameters or attributes

2.2.5.1 Using Low-Level I/O Primitives

You can alternatively enter I/O pin assignments using low-level I/O primitives. You can assign pin locations, I/O standards, drive strengths, slew rates, and on-chip termination (OCT) value assignments. You can also use low-level differential I/O primitives to define both positive and negative pins of a differential pair in the HDL code for your design.

Primitive-based assignments do not appear in the Pin Planner until after you perform a full compilation and back-annotate pin assignments (**Assignments > Back Annotate Assignments**).

Related Links

Designing with Low Level Primitives User Guide

2.3 Importing and Exporting I/O Pin Assignments

The Quartus Prime software supports transfer of I/O pin assignments across projects, or for analysis in third-party PCB tools. You can import or export I/O pin assignments in the following ways:

Table 4. Importing and Exporting I/O Pin Assignments

	Import Assignments	Export Assignments
Scenario	• From your PCB design tool or spreadsheet into Pin Planner during early pin planning or after optimization in PCB tool • From another Quartus Prime project with common constraints	• From Quartus Prime project for optimization in a PCB design tool • From Quartus Prime project for spreadsheet analysis or use in scripting assignments • From Quartus Prime project for import into another Quartus Prime project with similar constraints
Command	**Assignments ➤ Import Assignments**	**Assignments ➤ Export Assignments**
File formats	`.qsf`, `.esf`, `.acf`, `.csv`, `.txt`,`.sdc`	`.pin`,`.fx`,`.csv`, `.tcl`, `.qsf`
Notes	N/A	Exported `.csv` files retain column and row order and format. Do not modify the row of column headings if importing the `.csv` file

2.3.1 Importing and Exporting for PCB Tools

The Pin Planner supports import and export of assignments with PCB tools. You can export valid assignments as a **.pin** file for analysis in other supported PCB tools. You can also import optimized assignment from supported PCB tools. The **.pin** file contains pin name, number, and detailed properties.

Mentor Graphics I/O Designer requires you to generate and import both an **.fx** and a **.pin** file to transfer assignments. However, the Quartus Prime software requires only the **.fx** to import pin assignments from I/O Designer.

Table 5. Contents of .pin File

File Column Name	Description
Pin Name/Usage	The name of the design pin, or whether the pin is GND or V_{CC} pin
Location	The pin number of the location on the device package
Dir	The direction of the pin
I/O Standard	The name of the I/O standard to which the pin is configured
Voltage	The voltage level that is required to be connected to the pin
I/O Bank	The I/O bank to which the pin belongs
User Assignment	Y or N indicating if the location assignment for the design pin was user assigned (Y) or assigned by the Fitter (N)

Related Links

- Pin-Out Files for Intel Devices
- Mentor Graphics PCB Design Tools Support on page 279
  You can integrate the Mentor Graphics ® I/O Designer or DxDesigner PCB design tools into the Quartus Prime design flow. This combination provides a complete FPGA-to-board design workflow.

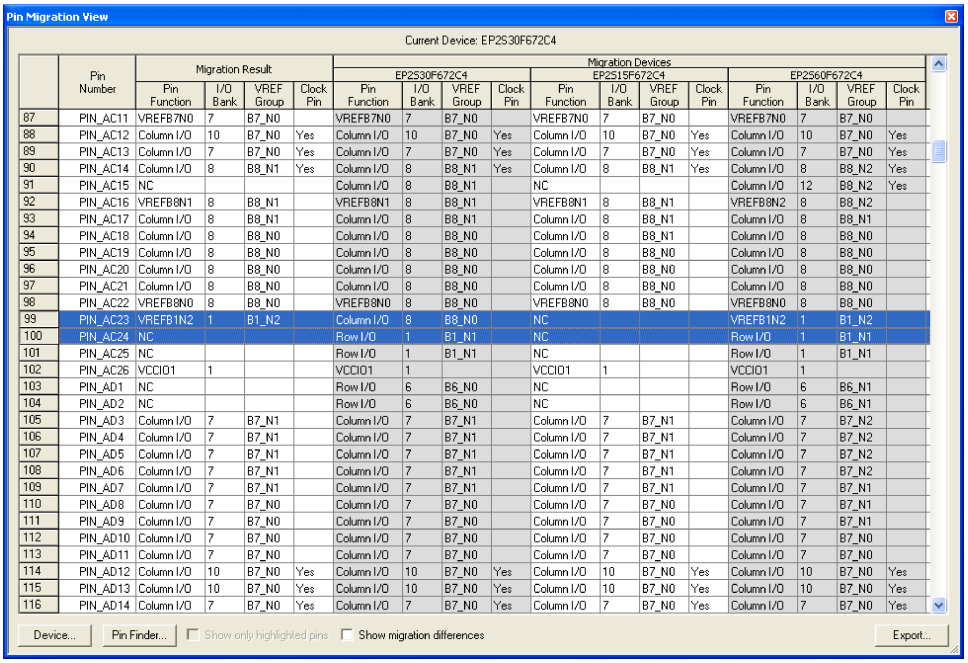
2.3.2 Migrating Assignments to Another Target Device

Click **View ➤ Pin Migration Window** to verify whether your pin assignments are compatible with migration to a different Intel device.

You can migrate compatible pin assignments from one target device to another. You can migrate to a different density and the same device package. You can also migrate between device packages with different densities and pin counts.

The Quartus Prime software ignores invalid assignments and generates an error message during compilation. After evaluating migration compatibility, modify any incompatible assignments, and then click **Export** to export the assignments to another project.

Figure 5. Device Migration Compatibility (AC24 does not exist in migration device)



	Pin Number	Migration Result				EP2S30F672C4				EP2S15F672C4				EP2S60F672C4			
		Pin Function	I/O Bank	VREF Group	Clock Pin	Pin Function	I/O Bank	VREF Group	Clock Pin	Pin Function	I/O Bank	VREF Group	Clock Pin	Pin Function	I/O Bank	VREF Group	Clock Pin
87	PIN_AC11	VREFB7N0	7	B7_N0		VREFB7N0	7	B7_N0		VREFB7N0	7	B7_N0		VREFB7N0	7	B7_N0	
88	PIN_AC12	Column I/O	10	B7_N0	Yes	Column I/O	10	B7_N0	Yes	Column I/O	10	B7_N0	Yes	Column I/O	10	B7_N0	Yes
89	PIN_AC13	Column I/O	7	B7_N0	Yes	Column I/O	7	B7_N0	Yes	Column I/O	7	B7_N0	Yes	Column I/O	7	B7_N0	Yes
90	PIN_AC14	Column I/O	8	B8_N1	Yes	Column I/O	8	B8_N1	Yes	Column I/O	8	B8_N1	Yes	Column I/O	8	B8_N2	Yes
91	PIN_AC15	NC				Column I/O	8	B8_N1		NC				Column I/O	12	B8_N2	Yes
92	PIN_AC16	VREFB8N1	8	B8_N1		VREFB8N1	8	B8_N1		VREFB8N1	8	B8_N1		VREFB8N2	8	B8_N2	
93	PIN_AC17	Column I/O	8	B8_N1		Column I/O	8	B8_N1		Column I/O	8	B8_N1		Column I/O	8	B8_N1	
94	PIN_AC18	Column I/O	8	B8_N0		Column I/O	8	B8_N0		Column I/O	8	B8_N1		Column I/O	8	B8_N0	
95	PIN_AC19	Column I/O	8	B8_N0		Column I/O	8	B8_N0		Column I/O	8	B8_N0		Column I/O	8	B8_N0	
96	PIN_AC20	Column I/O	8	B8_N0		Column I/O	8	B8_N0		Column I/O	8	B8_N0		Column I/O	8	B8_N0	
97	PIN_AC21	Column I/O	8	B8_N0		Column I/O	8	B8_N0		Column I/O	8	B8_N0		Column I/O	8	B8_N0	
98	PIN_AC22	VREFB8N0	8	B8_N0		VREFB8N0	8	B8_N0		VREFB8N0	8	B8_N0		VREFB8N0	8	B8_N0	
99	PIN_AC23	VREFB1N2	1	B1_N2		Column I/O	8	B8_N0		NC				VREFB1N2	1	B1_N2	
100	PIN_AC24	NC				Row I/O	1	B1_N1		NC				Row I/O	1	B1_N1	
101	PIN_AC25	NC				Row I/O	1	B1_N1		NC				Row I/O	1	B1_N1	
102	PIN_AC26	VCCIO1	1			VCCIO1	1			VCCIO1	1			VCCIO1	1		
103	PIN_AD1	NC				Row I/O	6	B6_N0		NC				Row I/O	6	B6_N1	
104	PIN_AD2	NC				Row I/O	6	B6_N0		NC				Row I/O	6	B6_N1	
105	PIN_AD3	Column I/O	7	B7_N1		Column I/O	7	B7_N1		Column I/O	7	B7_N1		Column I/O	7	B7_N2	
106	PIN_AD4	Column I/O	7	B7_N1		Column I/O	7	B7_N1		Column I/O	7	B7_N1		Column I/O	7	B7_N2	
107	PIN_AD5	Column I/O	7	B7_N1		Column I/O	7	B7_N1		Column I/O	7	B7_N1		Column I/O	7	B7_N2	
108	PIN_AD6	Column I/O	7	B7_N1		Column I/O	7	B7_N1		Column I/O	7	B7_N1		Column I/O	7	B7_N2	
109	PIN_AD7	Column I/O	7	B7_N1		Column I/O	7	B7_N1		Column I/O	7	B7_N1		Column I/O	7	B7_N1	
110	PIN_AD8	Column I/O	7	B7_N0		Column I/O	7	B7_N0		Column I/O	7	B7_N0		Column I/O	7	B7_N1	
111	PIN_AD9	Column I/O	7	B7_N0		Column I/O	7	B7_N0		Column I/O	7	B7_N0		Column I/O	7	B7_N1	
112	PIN_AD10	Column I/O	7	B7_N0		Column I/O	7	B7_N0		Column I/O	7	B7_N0		Column I/O	7	B7_N0	
113	PIN_AD11	Column I/O	7	B7_N0		Column I/O	7	B7_N0		Column I/O	7	B7_N0		Column I/O	7	B7_N0	
114	PIN_AD12	Column I/O	10	B7_N0	Yes	Column I/O	10	B7_N0	Yes	Column I/O	10	B7_N0	Yes	Column I/O	10	B7_N0	Yes
115	PIN_AD13	Column I/O	10	B7_N0	Yes	Column I/O	10	B7_N0	Yes	Column I/O	10	B7_N0	Yes	Column I/O	10	B7_N0	Yes
116	PIN_AD14	Column I/O	7	B7_N0	Yes	Column I/O	7	B7_N0	Yes	Column I/O	7	B7_N0	Yes	Column I/O	7	B7_N0	Yes

The migration result for the pin function of highlighted PIN_AC23 is not an NC but a voltage reference VREFB1N2 even though the pin is an NC in the migration device. VREF standards have a higher priority than an NC, thus the migration result display the voltage reference. Even if you do not use that pin for a port connection in your design, you must use the VREF standard for I/O standards that require it on the actual board for the migration device.

If one of the migration devices has pins intended for connection to V_{CC} or GND and these same pins are I/O pins on a different device in the migration path, the Quartus Prime software ensures these pins are not used for I/O. Ensure that these pins are connected to the correct PCB plane.

When migrating between two devices in the same package, pins that are not connected to the smaller die may be intended to connect to V_{CC} or GND on the larger die. To facilitate migration, you can connect these pins to V_{CC} or GND in your original design because the pins are not physically connected to the smaller die.

Related Links

AN90: SameFrame PinOut Design for FineLine BGA Packages

2.4 Validating Pin Assignments

The Quartus Prime software validates I/O pin assignments against predefined I/O rules for your target device. You can use the following tools to validate your I/O pin assignments throughout the pin planning process:

Table 6. I/O Validation Tools

I/O Validation Tool	Description	Click to Run
Advanced I/O Timing	Fully validates I/O assignments against all I/O and timing checks during compilation	**Processing ➤ Start Compilation**

2.4.1 I/O Assignment Validation Rules

I/O Assignment Analysis validates your assignments against the following rules:

Table 7. Examples of I/O Rule Checks

Rule	Description	HDL Required?
I/O bank capacity	Checks the number of pins assigned to an I/O bank against the number of pins allowed in the I/O bank.	No
I/O bank VCCIO voltage compatibility	Checks that no more than one VCCIO is required for the pins assigned to the I/O bank.	No
I/O bank VREF voltage compatibility	Checks that no more than one VREF is required for the pins assigned to the I/O bank.	No
I/O standard and location conflicts	Checks whether the pin location supports the assigned I/O standard.	No
I/O standard and signal direction conflicts	Checks whether the pin location supports the assigned I/O standard and direction. For example, certain I/O standards on a particular pin location can only support output pins.	No
Differential I/O standards cannot have open drain turned on	Checks that open drain is turned off for all pins with a differential I/O standard.	No
I/O standard and drive strength conflicts	Checks whether the drive strength assignments are within the specifications of the I/O standard.	No
Drive strength and location conflicts	Checks whether the pin location supports the assigned drive strength.	No
BUSHOLD and location conflicts	Checks whether the pin location supports BUSHOLD. For example, dedicated clock pins do not support BUSHOLD.	No

continued...

Rule	Description	HDL Required?
WEAK_PULLUP and location conflicts	Checks whether the pin location supports WEAK_PULLUP (for example, dedicated clock pins do not support WEAK_PULLUP).	No
Electromigration check	Checks whether combined drive strength of consecutive pads exceeds a certain limit. For example, the total current drive for 10 consecutive pads on a Stratix® II device cannot exceed 200 mA.	No
PCI_IO clamp diode, location, and I/O standard conflicts	Checks whether the pin location along with the I/O standard assigned supports PCI_IO clamp diode.	No
SERDES and I/O pin location compatibility check	Checks that all pins connected to a SERDES in your design are assigned to dedicated SERDES pin locations.	Yes
PLL and I/O pin location compatibility check	Checks whether pins connected to a PLL are assigned to the dedicated PLL pin locations.	Yes

Table 8. Signal Switching Noise Rules

Rule	Description	HDL Required?
I/O bank can not have single-ended I/O when DPA exists	Checks that no single-ended I/O pin exists in the same I/O bank as a DPA.	No
A PLL I/O bank does not support both a single-ended I/O and a differential signal simultaneously	Checks that there are no single-ended I/O pins present in the PLL I/O Bank when a differential signal exists.	No
Single-ended output is required to be a certain distance away from a differential I/O pin	Checks whether single-ended output pins are a certain distance away from a differential I/O pin.	No
Single-ended output has to be a certain distance away from a VREF pad	Checks whether single-ended output pins are a certain distance away from a VREF pad.	No
Single-ended input is required to be a certain distance away from a differential I/O pin	Checks whether single-ended input pins are a certain distance away from a differential I/O pin.	No
Too many outputs or bidirectional pins in a VREFGROUP when a VREF is used	Checks that there are no more than a certain number of outputs or bidirectional pins in a VREFGROUP when a VREF is used.	No
Too many outputs in a VREFGROUP	Checks whether too many outputs are in a VREFGROUP.	No

2.4.2 Running I/O Assignment Analysis

I/O assignment analysis validates I/O assignments against the complete set of I/O system and board layout rules. Full I/O assignment analysis validates blocks that directly feed or are fed by resources such as a PLL, LVDS, or gigabit transceiver blocks. In addition, the checker validates the legality of proper VREF pin use, pin locations, and acceptable mixed I/O standards

Run I/O assignment analysis during early pin planning to validate initial reserved pin assignments before compilation. Once you define design files, run I/O assignment analysis to perform more thorough legality checks with respect to the synthesized netlist. Run I/O assignment analysis whenever you modify I/O assignments.

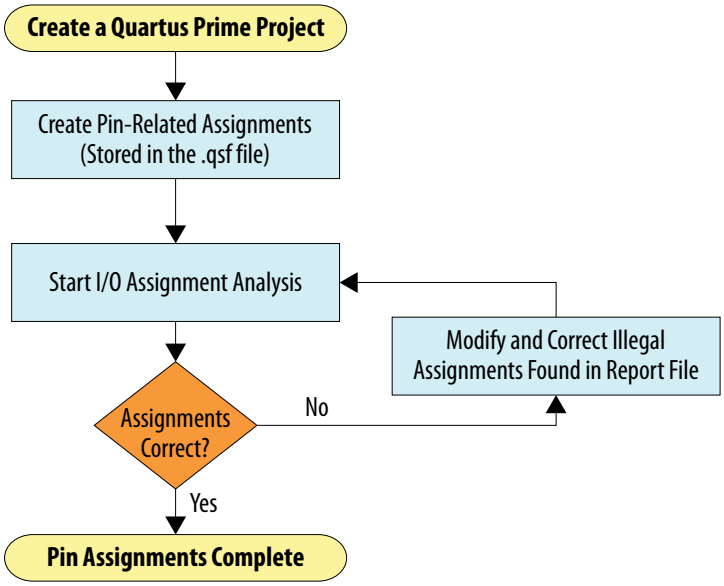
The Fitter assigns pins to accommodate your constraints. For example, if you assign an edge location to a group of LVDS pins, the Fitter assigns pin locations for each LVDS pin in the specified edge location and then performs legality checks. To display the Fitter-placed pins, click **Show Fitter Placements** in the Pin Planner. To accept these suggested pin locations, you must back-annotate your pin assignments.

View the I/O Assignment Warnings report to view and resolve all assignment warnings. For example, a warning that some design pins have undefined drive strength or slew rate. The Fitter recognizes undefined, single-ended output and bidirectional pins as non-calibrated OCT. To resolve the warning, assign the **Current Strength**, **Slew Rate** or **Slow Slew Rate** for the reported pin. Alternatively, you could assign the **Termination** to the pin. You cannot assign drive strength or slew rate settings when a pin has an OCT assignment.

2.4.2.1 Running Early I/O Assignment Analysis (Without Design Files)

You can perform basic I/O legality checks before defining HDL design files. This technique produces a preliminary board layout. For example, you can specify a target device and enter pin assignments that correspond to PCB characteristics. You can reserve and assign an I/O standards to each pin, and then run I/O assignment analysis to ensure that there are no I/O standard conflicts in each I/O bank.

Figure 6. Assigning and Analyzing Pin-Outs without Design Files



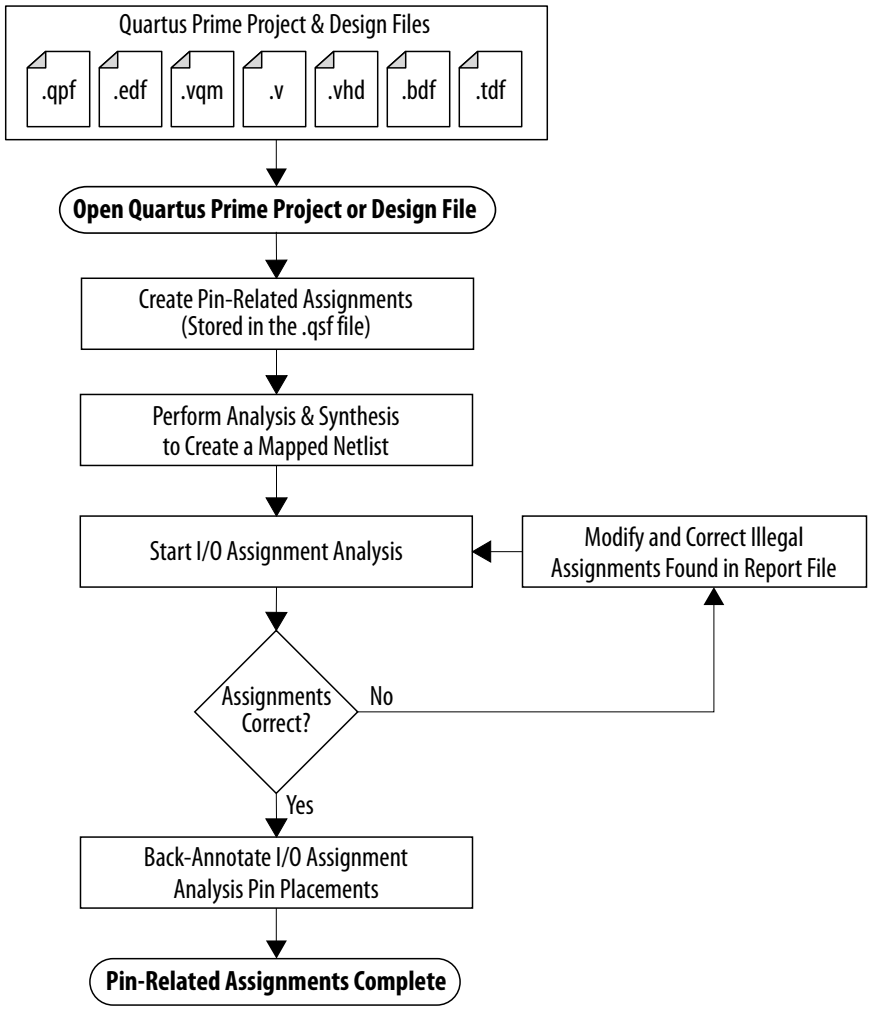
You must reserve all pins you intend to use as I/O pins, so that the Fitter can determine each pin type. After performing I/O assignment analysis, correct any errors reported by the Fitter and rerun I/O assignment analysis until all errors are corrected. A complete I/O assignment analysis requires all design files.

2.4.2.2 Running I/O Assignment Analysis (with Design Files)

Use I/O assignment analysis to perform full I/O legality checks after fully defining HDL design files. When you run I/O assignment analysis on a complete design, the tool verifies all I/O pin assignments against all I/O rules. When you run I/O assignment

analysis on a partial design, the tool checks legality only for defined portions of the design. The following figure shows the work flow for analyzing pin-outs with design files.

Figure 7. I/O Assignment Analysis Flow

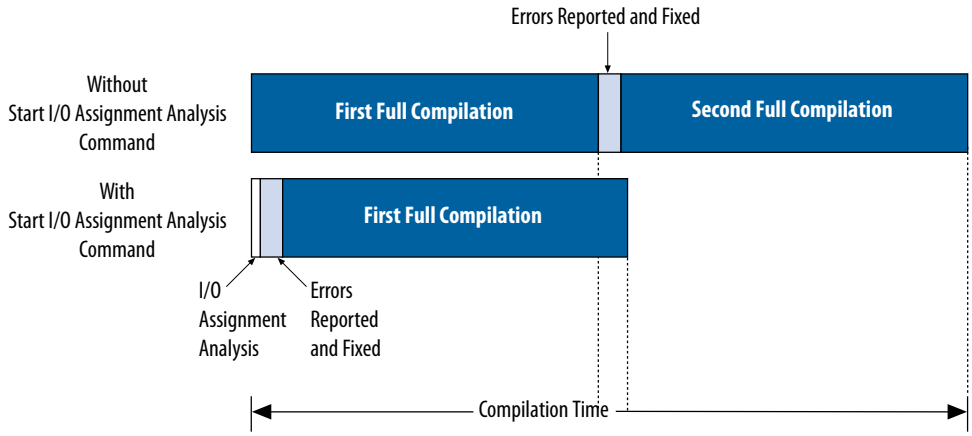


Even if I/O assignment analysis passes on incomplete design files, you may still encounter errors during full compilation. For example, you can assign a clock to a user I/O pin instead of assigning it to a dedicated clock pin, or design the clock to drive a PLL that you have not yet instantiated in the design. This occurs because I/O assignment analysis does not account for the logic that the pin drives, and does not verify that only dedicated clock inputs can drive the a PLL clock port.

To obtain better coverage, analyze as much of the design as possible over time, especially logic that connects to pins. For example, if your design includes PLLs or LVDS blocks, define these files prior to full analysis. After performing I/O assignment analysis, correct any errors reported by the Fitter and rerun I/O assignment analysis until all errors are corrected.

The following figure shows the compilation time benefit of performing I/O assignment analysis before running a full compilation.

Figure 8. I/O Assignment Analysis Reduces Compilation Time



2.4.2.3 Overriding Default I/O Pin Analysis

You can override the default I/O analysis of various pins to accommodate I/O rule exceptions, such as for analyzing VREF or inactive pins.

Each device contains a number of VREF pins, each supporting a number of I/O pins. A VREF pin and its I/O pins comprise a VREF bank. The VREF pins are typically assigned inputs with VREF I/O standards, such as HSTL- and SSTL-type I/O standards. Conversely, VREF outputs do not require the VREF pin. When a voltage-referenced input is present in a VREF bank, only a certain number of outputs can be present in that VREF bank. I/O assignment analysis treats bidirectional signals controlled by different output enables as independent output enables.

To assign the **Output Enable Group** option to bidirectional signals to analyze the signals as a single output enable group, follow these steps:

1. To access this assignment in the Pin Planner, right-click the **All pins** list and click **Customize Columns**.
2. Under **Available columns**, add **Output Enable Group** to **Show these columns in this order**. The column appears in the **All Pins** list.
3. Enter the same integer value for the **Output Enable Group** assignment for all sets of signals that are driving in the same direction.

This assignment is especially important for external memory interfaces. For example, consider a DDR2 interface in a Stratix II device. The device allows 30 pins in a VREF group. Each byte lane for a ×8 DDR2 interface includes one DQS pin and eight DQ pins, for a total of nine pins per byte lane. The DDR2 interface uses the **SSTL 18 Class I** VREF I/O standard. In typical interfaces, each byte lane has its own output enable. In this example, the DDR2 interface has four byte lanes. Using 30 I/O pins in a VREF group, there are three byte lanes and an extra byte lane that supports the three remaining pins. Without the **Output Enable Group** assignment, the Fitter analyzes each byte lane as an independent group driven by a unique output enable. In this worst-case scenario the three pins are inputs, and the other 27 pins are outputs violating the 20 output pin limit.

Because DDR2 DQS and DQ pins are always driven in the same direction, the analysis reports an error that is not applicable to your design. The **Output Enable Group** assignment designates the DQS and DQ pins as a single group driven by a common

output enable for I/O assignment analysis. When you use the **Output Enable Group** logic option assignment, the DQS and DQ pins are checked as all input pins or all output pins and are not in violation of the I/O rules.

You can also use the **Output Enable Group** assignment to designate pins that are driven only at certain times. For example, the data mask signal in DDR2 interfaces is an output signal, but it is driven only when the DDR2 is writing (bidirectional signals are outputs). To avoid I/O assignment analysis errors, use the **Output Enable Group** logic option assignment to assign the data mask to the same value as the DQ and DQS signals.

You can also use the **Output Enable Group** to designate VREF input pins that are inactive during the time the outputs are driving. This assignment removes the VREF input pins from the VREF analysis. For example, the QVLD signal for an RLDRAM II interface is active only during a read. During a write, the QVLD pin is not active and does not count as an active VREF input pin in the VREF group. Place the QVLD pins in the same output enable group as the RLDRAM II data pins.

Related Links

The TimeQuest Timing Analyzer
In *Quartus Prime Pro Edition Handbook Volume 3*

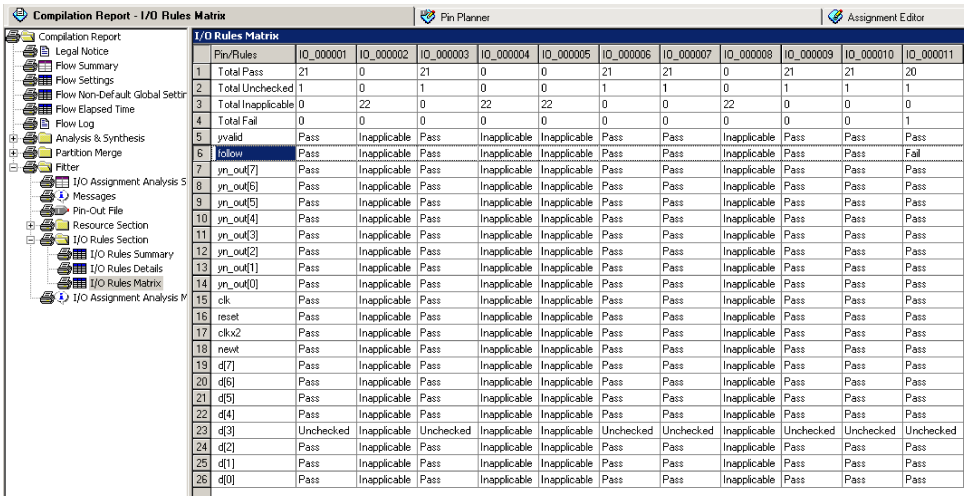
2.4.3 Understanding I/O Analysis Reports

The detailed I/O assignment analysis reports include the affected pin name and a problem description. The Fitter section of the Compilation report contains information generated during I/O assignment analysis, including the following reports:

- I/O Assignment Warnings—lists warnings generated for each pin
- Resource Section—quantifies use of various pin types and I/O banks
- I/O Rules Section—lists summary, details, and matrix information about the I/O rules tested

The **Status** column indicates whether rules passed, failed, or could not be checked. A severity rating indicates the rule's importance for effective analysis. "Inapplicable" rules do not apply to the target device family.

Figure 9. I/O Rules Matrix



I/O Rules Matrix

	Pin/Rules	IO_000001	IO_000002	IO_000003	IO_000004	IO_000005	IO_000006	IO_000007	IO_000008	IO_000009	IO_000010	IO_000011
1	Total Pass	21	0	21	0	0	21	21	0	21	21	20
2	Total Unchecked	1	0	1	0	0	1	1	0	1	1	1
3	Total Inapplicable	0	22	0	22	22	0	0	22	0	0	0
4	Total Fail	0	0	0	0	0	0	0	0	0	0	1
5	yvalid	Pass	Inapplicable	Pass	Inapplicable	Inapplicable	Pass	Pass	Inapplicable	Pass	Pass	Pass
6	follow	Pass	Inapplicable	Pass	Inapplicable	Inapplicable	Pass	Pass	Inapplicable	Pass	Pass	Fail
7	yn_out[7]	Pass	Inapplicable	Pass	Inapplicable	Inapplicable	Pass	Pass	Inapplicable	Pass	Pass	Pass
8	yn_out[6]	Pass	Inapplicable	Pass	Inapplicable	Inapplicable	Pass	Pass	Inapplicable	Pass	Pass	Pass
9	yn_out[5]	Pass	Inapplicable	Pass	Inapplicable	Inapplicable	Pass	Pass	Inapplicable	Pass	Pass	Pass
10	yn_out[4]	Pass	Inapplicable	Pass	Inapplicable	Inapplicable	Pass	Pass	Inapplicable	Pass	Pass	Pass
11	yn_out[3]	Pass	Inapplicable	Pass	Inapplicable	Inapplicable	Pass	Pass	Inapplicable	Pass	Pass	Pass
12	yn_out[2]	Pass	Inapplicable	Pass	Inapplicable	Inapplicable	Pass	Pass	Inapplicable	Pass	Pass	Pass
13	yn_out[1]	Pass	Inapplicable	Pass	Inapplicable	Inapplicable	Pass	Pass	Inapplicable	Pass	Pass	Pass
14	yn_out[0]	Pass	Inapplicable	Pass	Inapplicable	Inapplicable	Pass	Pass	Inapplicable	Pass	Pass	Pass
15	clk	Pass	Inapplicable	Pass	Inapplicable	Inapplicable	Pass	Pass	Inapplicable	Pass	Pass	Pass
16	reset	Pass	Inapplicable	Pass	Inapplicable	Inapplicable	Pass	Pass	Inapplicable	Pass	Pass	Pass
17	clkx2	Pass	Inapplicable	Pass	Inapplicable	Inapplicable	Pass	Pass	Inapplicable	Pass	Pass	Pass
18	newt	Pass	Inapplicable	Pass	Inapplicable	Inapplicable	Pass	Pass	Inapplicable	Pass	Pass	Pass
19	d[7]	Pass	Inapplicable	Pass	Inapplicable	Inapplicable	Pass	Pass	Inapplicable	Pass	Pass	Pass
20	d[6]	Pass	Inapplicable	Pass	Inapplicable	Inapplicable	Pass	Pass	Inapplicable	Pass	Pass	Pass
21	d[5]	Pass	Inapplicable	Pass	Inapplicable	Inapplicable	Pass	Pass	Inapplicable	Pass	Pass	Pass
22	d[4]	Pass	Inapplicable	Pass	Inapplicable	Inapplicable	Pass	Pass	Inapplicable	Pass	Pass	Pass
23	d[3]	Unchecked	Inapplicable	Unchecked	Inapplicable	Inapplicable	Unchecked	Unchecked	Inapplicable	Unchecked	Unchecked	Unchecked
24	d[2]	Pass	Inapplicable	Pass	Inapplicable	Inapplicable	Pass	Pass	Inapplicable	Pass	Pass	Pass
25	d[1]	Pass	Inapplicable	Pass	Inapplicable	Inapplicable	Pass	Pass	Inapplicable	Pass	Pass	Pass
26	d[0]	Pass	Inapplicable	Pass	Inapplicable	Inapplicable	Pass	Pass	Inapplicable	Pass	Pass	Pass

2.5 Verifying I/O Timing

You must verify board-level signal integrity and I/O timing when assigning I/O pins. High-speed interface operation requires a quality signal and low propagation delay at the far end of the board route. Click **Tools ➤ TimeQuest Timing Analyzer** to confirm timing after making I/O pin assignments.

For example, if you change the slew rates or drive strengths of some I/O pins with ECOs, you can verify timing without recompiling the design. You must understand I/O timing and what factors affect I/O timing paths in your design. The accuracy of the output load specification of the output and bidirectional pins affects the I/O timing results.

The Quartus Prime software supports three different methods of I/O timing analysis:

Table 9. I/O Timing Analysis Methods

I/O Timing Analysis	Description
Advanced I/O timing analysis	Analyze I/O timing with your board trace model to report accurate, "board-aware" simulation models. Configures a complete board trace model for each I/O standard or pin. TimeQuest applies simulation results of the I/O buffer, package, and board trace model to generate accurate I/O delays and system level signal information. Use this information to improve timing and signal integrity.
I/O timing analysis	Analyze I/O timing with default or specified capacitive load without signal integrity analysis. TimeQuest reports tCO to an I/O pin using a default or user-specified value for a capacitive load.
Full board routing simulation	Use Intel-provided or Quartus Prime software-generated IBIS or HSPICE I/O models for simulation in Mentor Graphics HyperLynx and Synopsys HSPICE.

For more information about advanced I/O timing support, refer to the appropriate device handbook for your target device. For more information about board-level signal integrity and tips on how to improve signal integrity in your high-speed designs, refer to the Altera Signal Integrity Center page of the Altera website.

For information about creating IBIS and HSPICE models with the Quartus Prime software and integrating those models into HyperLynx and HSPICE simulations, refer to the *Signal Integrity Analysis with Third Party Tools* chapter.

Related Links

- Literature and Technical Documentation
- Altera Signal Integrity Center
- Signal Integrity Analysis with Third-Party Tools on page 221

2.5.1 Running Advanced I/O Timing

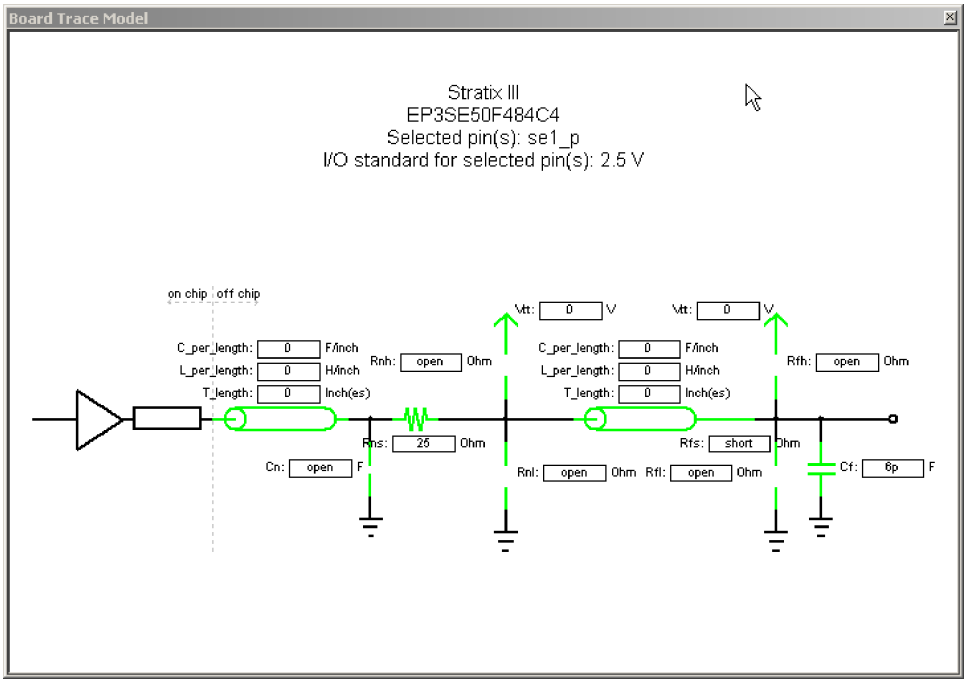
Advanced I/O timing analysis uses your board trace model and termination network specification to report accurate output buffer-to-pin timing estimates, FPGA pin and board trace signal integrity and delay values. Advanced I/O timing runs automatically for supported devices during compilation.

2.5.1.1 Understanding the Board Trace Models

The Quartus Prime software provides board trace model templates for various I/O standards. The following figure shows the template for a **2.5 V** I/O standard. This model consists of near-end and far-end board component parameters.

Near-end board trace modeling includes the elements which are close to the device. Far-end modeling includes the elements which are at the receiver end of the link, closer to the receiving device. Board trace model topology is conceptual and does not necessarily match the actual board trace for every component. For example, near-end model parameters can represent device-end discrete termination and breakout traces. Far-end modeling can represent the bulk of the board trace to discrete external memory components, and the far end termination network. You can analyze the same circuit with near-end modeling of the entire board, including memory component termination, and far-end modeling of the actual memory component.

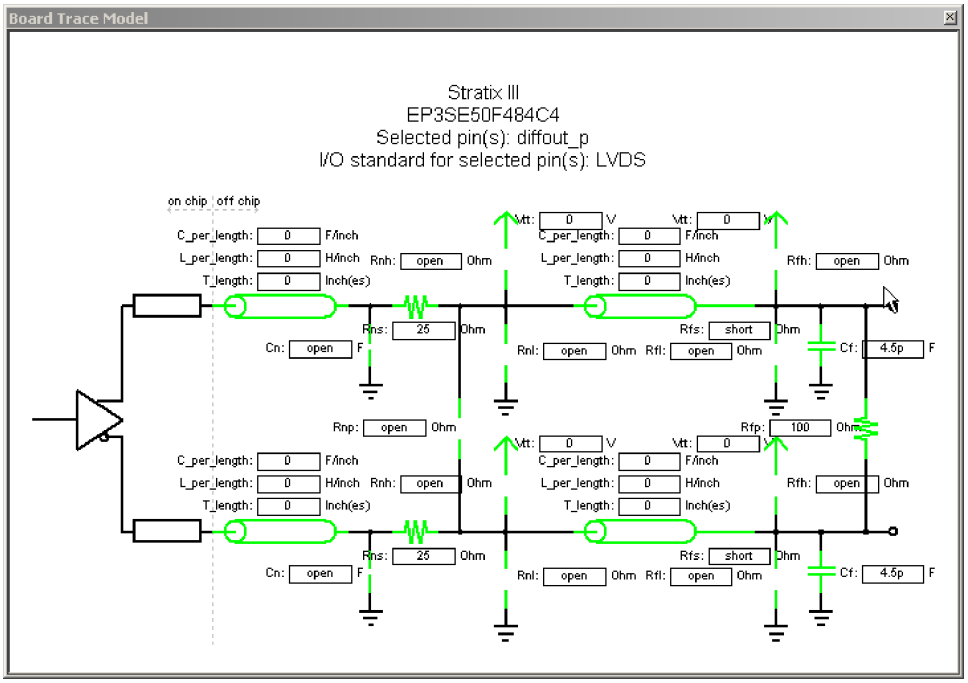
Figure 10. 2.5-V I/O Standard Board Trace Model



The following figure shows the template for the **LVDS** I/O standard. The far-end capacitance (Cf) represents the external-device or multiple-device capacitive load. If you have multiple devices on the far-end, you must find the equivalent capacitance at the far-end, taking into account all receiver capacitances. The far-end capacitance can be the sum of all the receiver capacitances.

The Quartus Prime software models lossless transmission lines, and does not require a transmission-line resistance value. Only distributed inductance (L) and capacitance (C) values are needed. The distributed L and C values of transmission lines must be entered on a per-inch basis, and can be obtained from the PCB vendor or manufacturer, the CAD Design tool, or a signal integrity tool, such as the Mentor Graphics Hyperlynx software.

Figure 11. LVDS Differential Board Trace Model



2.5.1.2 Defining the Board Trace Model

The board trace model describes a board trace and termination network as a set of capacitive, resistive, and inductive parameters.

Advanced I/O Timing uses the model to simulate the output signal from the output buffer to the far end of the board trace. You can define the capacitive load, any termination components, and trace impedances in the board routing for any output pin or bidirectional pin in output mode. You can configure an overall board trace model for each I/O standard or for specific pins. Define an overall board trace model for each I/O standard in your design. Use that model for all pins that use the I/O standard. You can customize the model for specific pins using the **Board Trace Model** window in the Pin Planner.

1. Click **Assignments ➤ Device ➤ Device and Pin Options**.
2. Click **Board Trace Model** and define board trace model values for each I/O standard.
3. Click **I/O Timing** and define default I/O timing options at board trace near and far ends.
4. Click **Assignments ➤ Pin Planner** and assign board trace model values to individual pins.

Example 3. Specifying Board Trace Model

```
## setting the near end series resistance model of sel_p output pin to 25 ohms
set_instance_assignment -name BOARD_MODEL_NEAR_SERIES_R 25 -to se1_p
## Setting the far end capacitance model for sel_p output signal to 6 picofarads
set_instance_assignment -name BOARD_MODEL_FAR_C 6P -to se1_p
```

Related Links

Board Trace Mode
In Quartus Prime Help

2.5.1.3 Modifying the Board Trace Model

To modify the board trace model, click **View ➤ Board Trace Model** in the Pin Planner.

You can modify any of the board trace model parameters within a graphical representation of the board trace model.

The **Board Trace Model** window displays the routing and components for positive and negative signals in a differential signal pair. Only modify the positive signal of the pair, as the setting automatically applies to the negative signal. Use standard unit prefixes such as *p*, *n*, and *k* to represent pico, nano, and kilo, respectively. Use the **short** or **open** value to designate a short or open circuit for a parallel component.

2.5.1.4 Specifying Near-End vs Far-End I/O Timing Analysis

You can select a near-end or far-end point for I/O timing analysis. Near-end timing analysis extends to the device pin. You can apply the `set_output_delay` constraint during near-end analysis to account for the delay across the board.

With far-end I/O timing analysis, the advanced I/O timing analysis extends to the external device input, at the far-end of the board trace. Whether you choose a near-end or far-end timing endpoint, the board trace models are taken into account during timing analysis.

2.5.1.5 Understanding Advanced I/O Timing Analysis Reports

View I/O timing analysis information in the following reports:

Table 10. Advanced I/O Timing Reports

I/O Timing Report	Description
TimeQuest Report	Reports signal integrity and board delay data.
Board Trace Model Assignments report	Summarizes the board trace model component settings for each output and bidirectional signal.
Signal Integrity Metrics report	Contains all the signal integrity metrics calculated during advanced I/O timing analysis based on the board trace model settings for each output or bidirectional pin. Includes measurements at both the FPGA pin and at the far-end load of board delay, steady state voltages, and rise and fall times.

Note: By default, the TimeQuest analyzer generates the Slow-Corner Signal Integrity Metrics report. To generate a Fast-Corner Signal Integrity Metrics report you must change the delay model by clicking **Tools ➤ TimeQuest Timing Analyzer**.

Related Links

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

2.5.2 Adjusting I/O Timing and Power with Capacitive Loading

When calculating t_{CO} and power for output and bidirectional pins, the TimeQuest analyzer and the Power Analyzer use a bulk capacitive load. You can adjust the value of the capacitive load per I/O standard to obtain more precise t_{CO} and power measurements, reflecting the behavior of the output or bidirectional net on your PCB. The Quartus Prime software ignores capacitive load settings on input pins. You can adjust the capacitive load settings per I/O standard, in picofarads (pF), for your entire design. During compilation, the Compiler measures power and t_{CO} measurements based on your settings. You can also adjust the capacitive load on an individual pin with the **Output Pin Load** logic option.

2.6 Viewing Routing and Timing Delays

Right-click any node and click **Locate > Locate in Chip Planner** to visualize and adjust I/O timing delays and routing between user I/O pads and V_{CC}, GND, and V_{REF} pads. The Chip Planner graphically displays logic placement, LogicLock® Plus regions, relative resource usage, detailed routing information, fan-in and fan-out, register paths, and high-speed transceiver channels. You can view physical timing estimates, routing congestion, and clock regions. Use the Chip Planner to change connections between resources and make post-compilation changes to logic cell and I/O atom placement. When you select items in the Pin Planner, the corresponding item is highlighted in Chip Planner.

2.7 Scripting API

You can alternatively use Tcl commands to access I/O management functions, rather than using the GUI. For detailed information about specific scripting command options and Tcl API packages, type the following command at a system command prompt to view the Tcl API Help browser:

```
quartus_sh --qhelp
```

Related Links

- Tcl Scripting on page 72
- Command Line Scripting on page 65
  FPGA design software that easily integrates into your design flow saves time and improves productivity. The Intel Quartus Prime software provides you with a command-line executable for each step of the FPGA design flow to make the design process customizable and flexible.

2.7.1 Generate Mapped Netlist

Enter the following in the Tcl console or in a Tcl script:

```
execute_module -tool map
```

The `execute_module` command is in the flow package.

Type the following at a system command prompt:

```
quartus_syn <project name>
```

2.7.2 Reserve Pins

Use the following Tcl command to reserve a pin:

```
set_instance_assignment -name RESERVE_PIN <value> -to <signal name>
```

Use one of the following valid reserved pin values:

- "AS BIDIRECTIONAL"
- "AS INPUT TRI STATED"
- "AS OUTPUT DRIVING AN UNSPECIFIED SIGNAL"
- "AS OUTPUT DRIVING GROUND"
- "AS SIGNALPROBE OUTPUT"

Note: You must include the quotation marks when specifying the reserved pin value.

2.7.3 Set Location

Use the following Tcl command to assign a signal to a pin or device location:

```
set_location_assignment <location> -to <signal name>
```

Valid locations are pin locations, I/O bank locations, or edge locations. Pin locations include pin names, such as PIN_A3. I/O bank locations include IOBANK_1 up to IOBANK_ *n*, where *n* is the number of I/O banks in the device.

Use one of the following valid edge location values:

- EDGE_BOTTOM
- EDGE_LEFT
- EDGE_TOP
- EDGE_RIGHT

2.7.4 Exclusive I/O Group

Use the following Tcl command to create an exclusive I/O group assignments:

```
set_instance_assignment -name "EXCLUSIVE_IO_GROUP" -to pin
```

2.7.5 Slew Rate and Current Strength

Use the following Tcl commands to create an slew rate and drive strength assignments:

```
set_instance_assignment -name CURRENT_STRENGTH_NEW 8MA -to e[0]
set_instance_assignment -name SLEW_RATE 2 -to e[0]
```

Related Links

Package Information Datasheet for Mature Altera Devices

2.8 Document Revision History

The following table shows the revision history for this chapter.

Table 11. Document Revision History

Date	Version	Changes
2017.05.08	17.0.0	• *Run I/O Assignment Analysis* command renamed to *Start Fitter (Plan)*
2016.10.31	16.1.0	• Implemented Intel rebranding.
2015.11.02	15.1.0	• Removed early pin planning and Live I/O Check support from Quartus Prime Pro Edition handbook • Changed instances of *Quartus II* to *Quartus Prime*.
2014.12.15	14.1.0	• Updated Live I/O check device support to include only limited device families.
2014.08.30	14.0a10.0	• Added link to information about special pin assignment features for Arria 10 SoC devices.
2014.06.30	14.0.0	• Replaced MegaWizard Plug-In Manager information with IP Catalog.
November 2013	13.1.0	• Reorganization and conversion to DITA.
May 2013	13.0.0	• Added information about overriding I/O placement rules.
November 2012	12.1.0	• Updated Pin Planner description for new task and report windows.
June 2012	12.0.0	• Removed survey link.
November 2011	11.1.0	• Minor updates and corrections. • Updated the document template.
December 2010	10.0.1	Template update
July 2010	10.0.0	• Reorganized and edited the chapter • Added links to Help for procedural information previously included in the chapter • Added information on rules marked Inapplicable in the I/O Rules Matrix Report • Added information on assigning slew rate and drive strength settings to pins to fix I/O assignment warnings
November 2009	9.1.0	• Reorganized entire chapter to include links to Help for procedural information previously included in the chapter • Added documentation on near-end and far-end advanced I/O timing
March 2009	9.0.0	• Updated “Pad View Window” on page 5–20 • Added new figures: • Figure 5–15 • Figure 5–16 • Added new section “Viewing Simultaneous Switching Noise (SSN) Results” on page 5–17 • Added new section “Creating Exclusive I/O Group Assignments” on page 5–18

Related Links

Altera Documentation Archive

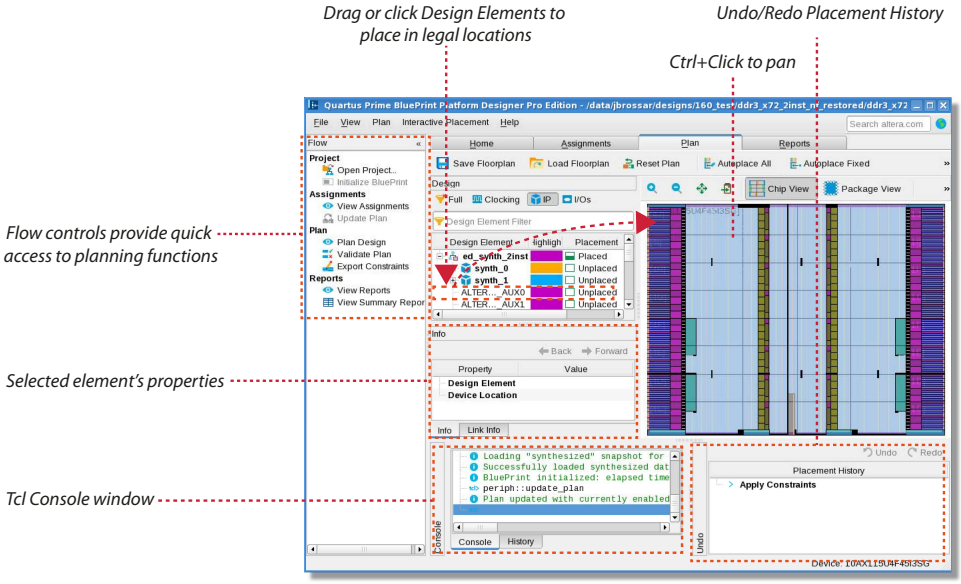
For previous versions of the *Quartus Prime Handbook*, search the Altera documentation archives.

3 BluePrint Design Planning

Design planning—the feasibility analysis of physical constraints—is a fundamental early step in advanced FPGA design. Periphery placement can be a complex process involving many variables. The BluePrint Platform Designer simplifies the planning of accurate constraints for physical implementation. Use BluePrint to prototype interface implementations, plan clocks, and rapidly define a legal device floorplan.

BluePrint interacts dynamically with the Quartus Prime Fitter to accurately verify placement legality while you plan. You can evaluate different floorplans, using interactive reports to accurately plan the best implementation without iterative compilation. Fitter verification ensures the highest correlation between your BluePrint plan and actual implementation results. You can apply the BluePrint plan constraints to your project with high confidence in the final implementation.

Figure 12. BluePrint Platform Designer GUI



BluePrint Platform Designer Features

BluePrint provides support for the following:

- Plan legal design periphery floorplans at any stage of the design process.
- Analyze and modify clock network scenarios to direct the placement of high fan-out signals.
- Automatically evaluate all placement legality using the Fitter.
- Save and reload various floorplan files.
- GUI and Tcl command-line operation.

Related Links

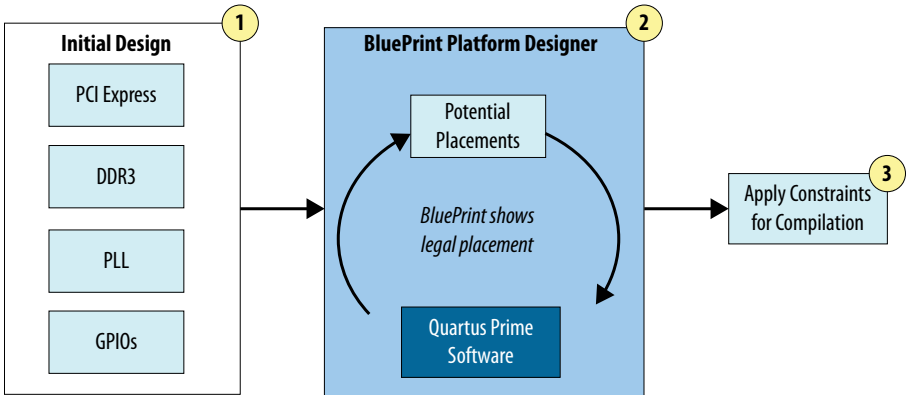
Video Demo: Using BluePrint to Place DDR-3 and PCI Express Gen3

3.1 BluePrint Planning Overview

After design synthesis, use BluePrint to rapidly define a legal device floorplan. BluePrint planning involves initialization of BluePrint, reconciliation of project assignments, placement of periphery elements and clocks, and export of plan constraints to your Quartus Prime project.

Note: The plan constraints that you define in BluePrint do not automatically apply to your Quartus Prime project until you add the BluePrint constraints to your project via a generated Tcl script.

Figure 13. BluePrint Streamlines Legal Placement



Intel FPGAs contain core and periphery device locations. The device core locations are adaptive look-up tables (ALUTs), core flip-flops, RAMs, and digital signal processors (DSPs). Device periphery locations include I/O elements, phase-locked loops (PLLs), clock buffers, and hard processor systems (HPS).

Intel's advanced FPGAs contain many silicon features in the periphery, such as hard PCI Express® IP cores, high speed transceivers, hard memory interface circuitry, and embedded processors. Interactions among these periphery atoms can be complex. BluePrint simplifies this complexity and allows you to quickly visualize and place I/O interface and periphery elements, such as:

- I/O elements
- LVDS interfaces
- PLLs
- Clocks
- Hard interface IP Cores
- High-Speed Transceivers
- Hard Memory Interface IP Cores
- Embedded Processors

After initialization, BluePrint displays your project's logical hierarchy, post-synthesis design elements, and Fitter-created design elements, alongside a view of target device locations. The GUI supports a variety of methods for placing design elements in the floorplan. As you place elements in the floorplan, the Fitter verifies legality in real time to ensure accurate correlation with the final implementation.

Related Links

Managing Device I/O Pins on page 18

3.2 Using BluePrint

BluePrint's user interface guides you through the design planning steps. You proceed through each BluePrint tab as you successively complete the setup, initialization, placement, and reporting steps. BluePrint allows you to move forward only after you complete the prerequisite steps.

Figure 14. BluePrint GUI

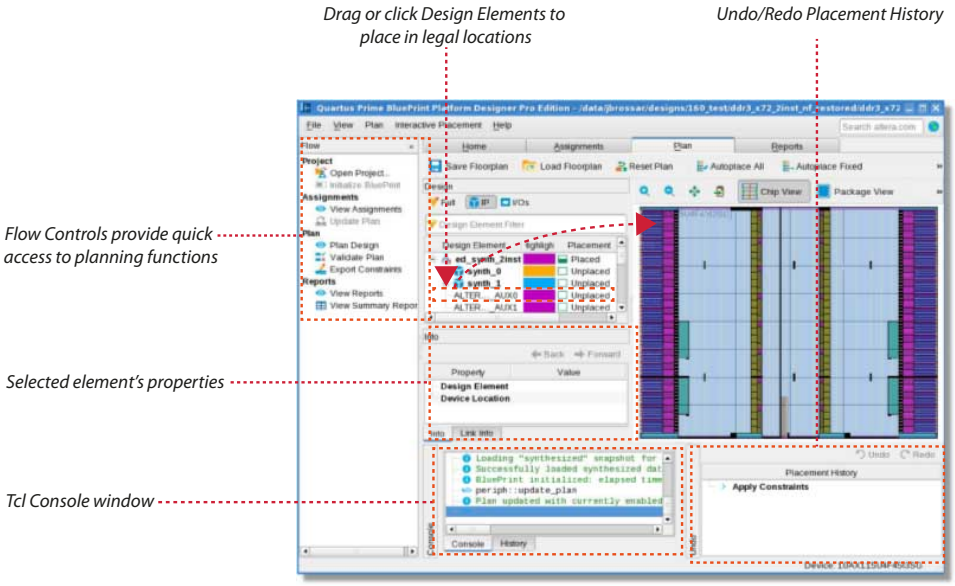
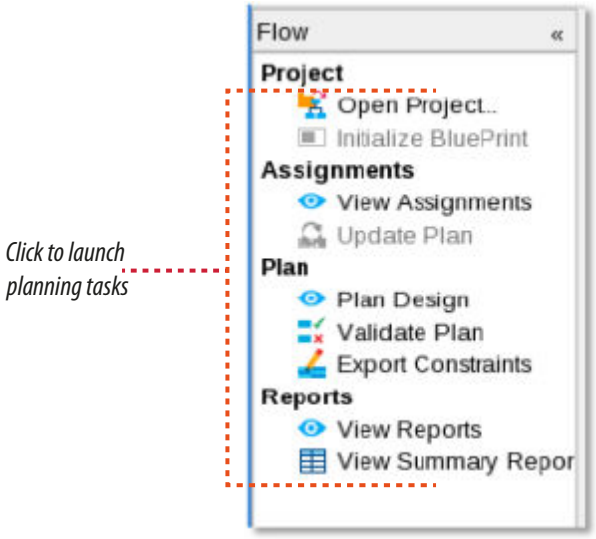


Table 12. BluePrint User Interface Reference

Use this BluePrint Tab	For This Task
Home	Load a project and initialize BluePrint.
Assignments	Resolve (disable) any conflicting assignments from your synthesized project. Click **Update Plan** to apply your enabled project assignments to the BluePrint plan.
Plan	Interactively plan placement with real-time validation and feedback in a graphical floorplan of the device.
Reports	Generate detailed, interactive reports to inform implementation decisions.
Tcl Console window	View and enter Tcl commands that control Blueprint user.
Undo/Redo Placement History	Display the history of placement operations in BluePrint. The **Undo** button reverts the last change made in the **Plan** tab. **Redo** re-implements the last **Undo** operation. Use these buttons to step forward and backward though your plan changes.

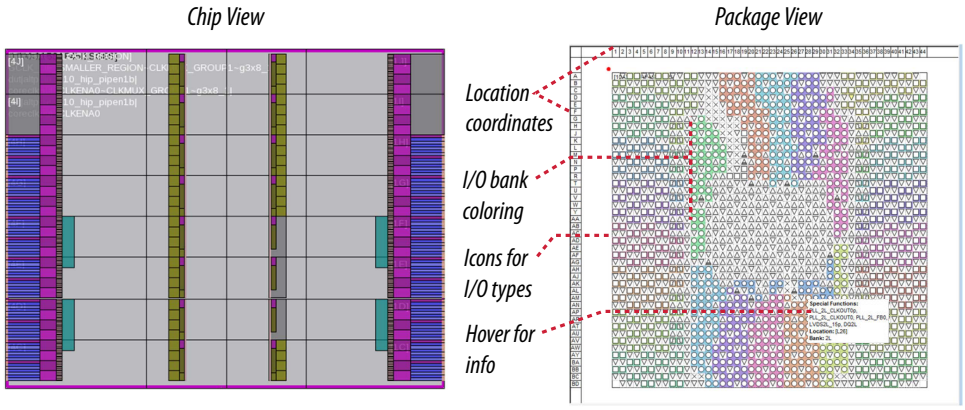
BluePrint's **Flow** control provides immediate access to the main tasks in the BluePrint planning flow. The **Flow** controls appear in order of a typical BluePrint planning flow. Click the **Flow** controls to complete the setup and initialization tasks in order.

Figure 15. BluePrint Flow Control



BluePrint accounts for any imported project assignments. You can disable or enable imported project assignments to resolve any conflicts and evaluate different implementations. After setup and initialization, you can place design elements onto the **Chip View** or **Package View** of the target device. During placement, BluePrint provides real-time, accurate feedback about legal placement options verified by the Fitter.

Figure 16. BluePrint Chip and Package Views



After you are satisfied with the BluePrint plan for important interfaces, you validate the plan to confirm that the Fitter can place all remaining logic. On validation, you can apply the BluePrint plan to your project by sourcing a Tcl file that BluePrint generates.

Step 1: Setup the Project on page 46
BluePrint requires at least a partially complete, synthesized Quartus Prime project as input. You can also use BluePrint to place a fully complete design project.

Step 2: Initialize BluePrint on page 46
To open a project in BluePrint:

Step 3: Update Plan with Project Assignments on page 47

Before periphery planning in BluePrint, you must reconcile any imported project assignments and **Update Plan** with the assignments.

Step 4: Plan Periphery Placement on page 48
Click **Plan Design** on the **Flow** control to interactively place IP cores and other design elements in legal locations in the device periphery.

Step 5: Report Placement Data on page 53
Generate BluePrint placement and connectivity reports to help locate cells and make the best decisions about placement for the interfaces and elements in your design.

Step 6: Validate and Export Plan Constraints on page 54
You must validate your BluePrint plan before exporting the plan constraints to your project as a generated Tcl script. Validation must confirm that the Fitter can place all remaining unplaced design elements.

3.2.1 Step 1: Setup the Project

BluePrint requires at least a partially complete, synthesized Quartus Prime project as input. You can also use BluePrint to place a fully complete design project. Before planning in BluePrint, prepare the Quartus Prime project in the following ways:

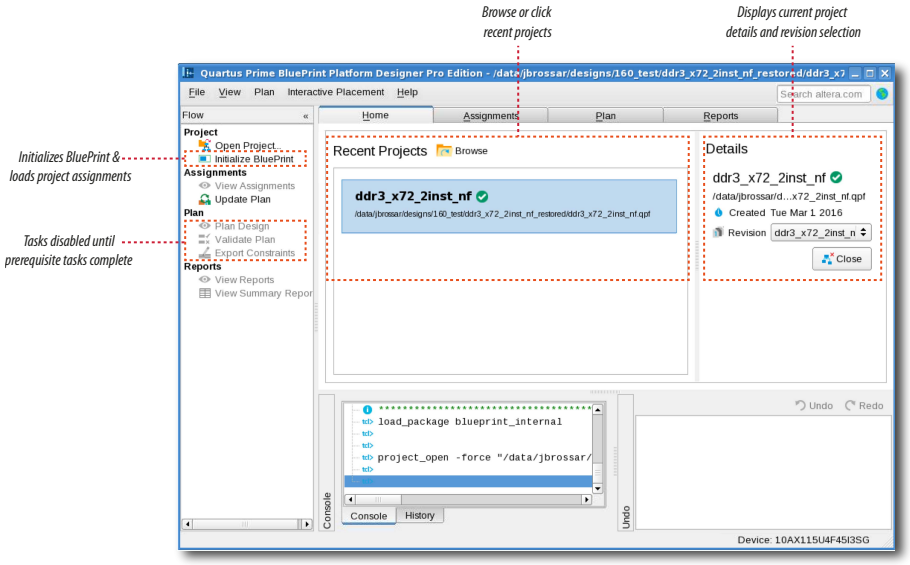
- Fully define known device periphery interfaces.
- Instantiate all known interface IP cores.
- Declare all general purpose I/Os.
- Define the I/O standard, voltage, drive strength, and slew rate for all general purpose I/Os.
- Define the core clocking (optional, but recommended).
- Connect all interfaces of the periphery IP to virtual pins or test logic. This technique creates loop backs on any interfaces in the shell design, helping to ensure that periphery interfaces persist after synthesis optimization.

3.2.2 Step 2: Initialize BluePrint

To open a project in BluePrint:

1. Click **Tools ➤ BluePrint Platform Designer**. The Quartus Prime software closes, and BluePrint opens, displaying the **Home** tab.
2. Click **Initialize BluePrint** on the **Flow** control to start the Fitter validation and import any project assignments. After initialization, the Fitter dynamically validates your BluePrint plan as you make changes.

Figure 17. BluePrint Home Tab



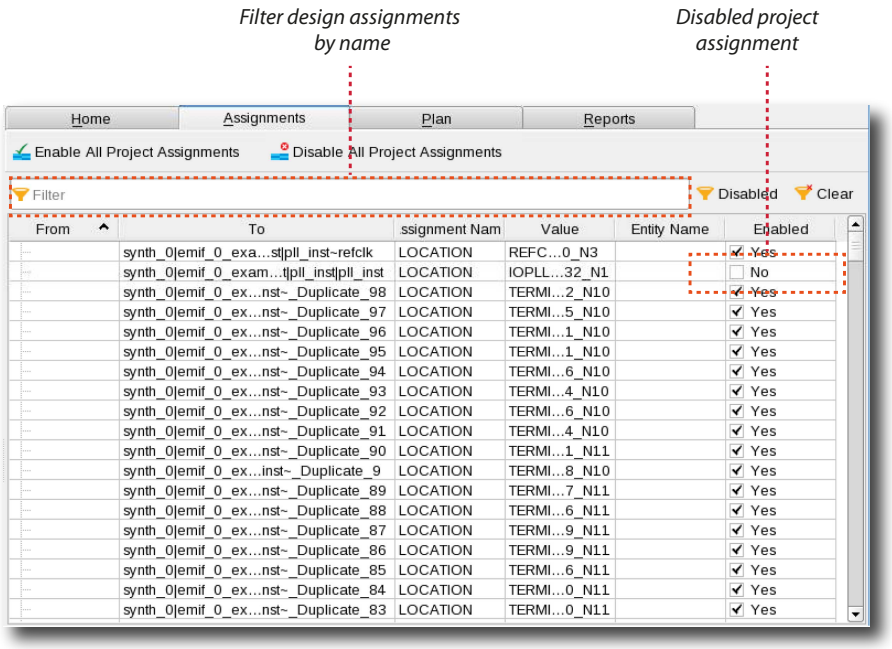
Note: The changes that you make in BluePrint do not apply to your Quartus Prime project until you export and apply the BluePrint plan constraints to your project.

3.2.3 Step 3: Update Plan with Project Assignments

Before periphery planning in BluePrint, you must reconcile any imported project assignments and **Update Plan** with the assignments.

1. Click **View Assignments** on the **Flow** control to review imported project assignments and reconcile any conflicts. You can filter the list of assignments by assignment name or status. Enable or disable specific or groups of project assignments to resolve any conflicts.
2. Click **Plan ➤ Reset Plan** if you want to return all project assignments to the enabled state.
3. After resolving all conflicts, click **Update Plan** on the **Flow** control to apply the enabled project assignments to your BluePrint plan.

Figure 18. BluePrint Platform Designer (Assignments Tab)



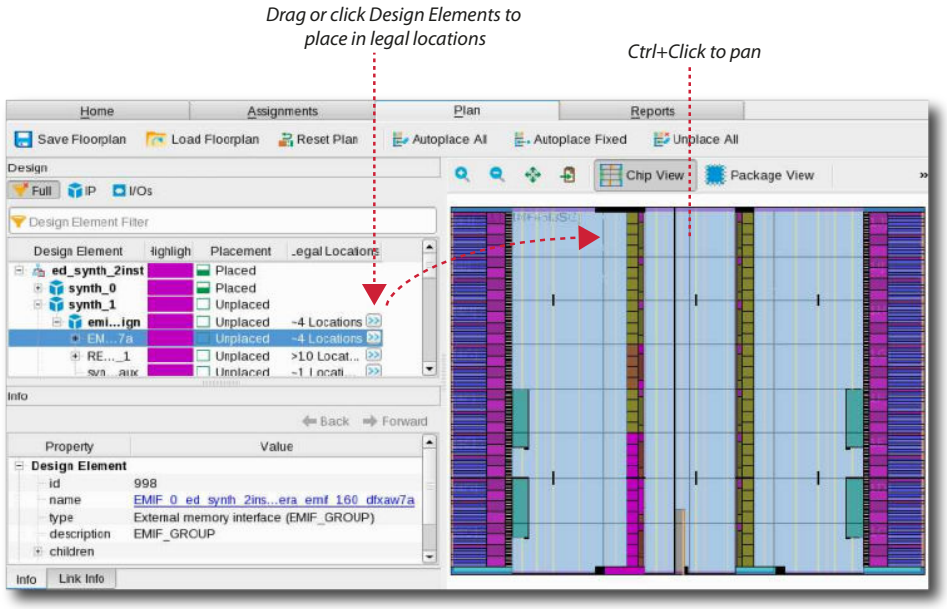
Related Links

- Home Tab Controls on page 55
  The BluePrint **Home** tab contains the following controls for opening projects in BluePrint.
- Assignments Tab Controls on page 55
  The **Assignments** tab contains controls for resolving potential conflicts with project assignments.

3.2.4 Step 4: Plan Periphery Placement

Click **Plan Design** on the **Flow** control to interactively place IP cores and other design elements in legal locations in the device periphery. The **Plan** tab displays a list of your project's design elements, alongside a graphical abstraction of the target device architecture.

Figure 19. BluePrint Platform Designer (Plan Tab)



Use the following controls to place design elements in the BluePrint floorplan:

- Drag elements from the **Design Elements** list and drop them onto available device resources in the **Chip** or **Package** view.
- Alternatively, click the >> button next to any design element to display a list of **Legal Locations**. Click any legal location in the list to highlight the location in the floorplan. Double-click any legal location in the list to place in the floorplan.
- Right-click and select **Autoplace Selected** to have BluePrint set placement for an unplaced design element. All unplaced clocks require that you use **Autoplace Selected**.
- Use the **Undo** and **Redo** buttons to step forward and backward though your plan changes.
- Use Ctrl+Click to drag and pan across the **Chip** or **Package** views.
- Use the **Link Info** tab to visualize and traverse design connectivity. For example, you can view the reference clock pin and driven destination cells of a PLL. Often these links have significant impact on placement flexibility because of dedicated routing paths for periphery features. Use the Back and Forward buttons to traverse design connectivity.

Figure 20. Listing Legal Locations

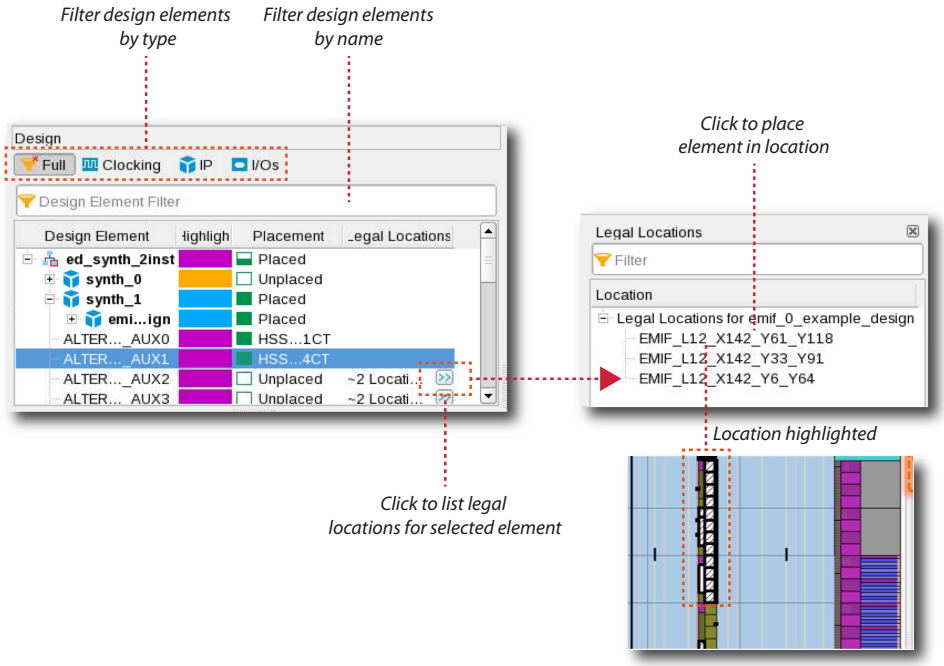
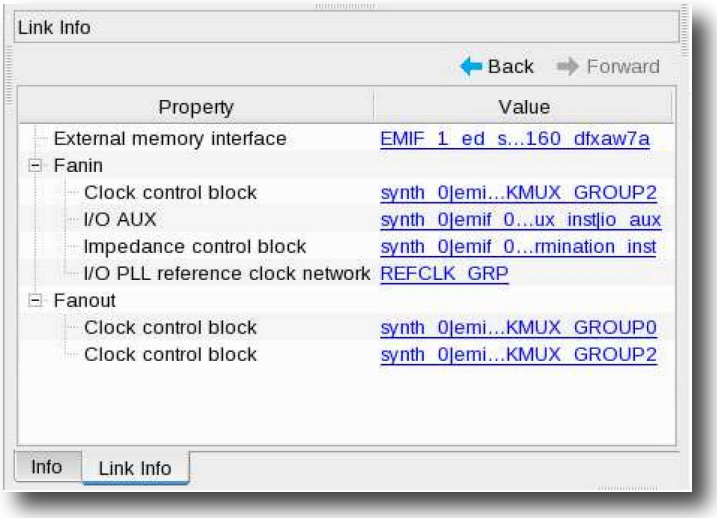
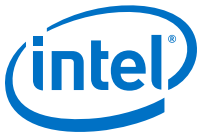


Figure 21. Link Info Tab for Traversing Connectivity





You can apply filters to locate specific design elements or target specific device resources for placement. Device resources are color coded by type, allowing quick access to any structure. Click the **Highlight** column to customize the color definitions. Place elements in the following order for efficiency:

1. Place all I/O pins or elements, such as PLLs, that have known, specific location requirements.
2. Place all known periphery interface IP.
3. (Optional) Place all remaining unplaced cells (**Autoplace Fixed** or right-click any individual unplaced element to **Autoplace Selected Element**).

Select any physical design element and click **Report Placeability of Selected Element** to generate a report that shows the placement locations the Fitter prefers. Right-click reported locations to directly place cells in the location. Show error messages for unsupported locations to help understand the device constraints. You can **Export** the report content for analysis in other tools.

Note: Changes made in BluePrint do not apply to your Quartus Prime project until you apply the generated BluePrint plan constraints script to your project.

Related Links

Plan Tab Controls on page 56
The **Plan** tab contains the following controls to help you place logic in the BluePrint plan.

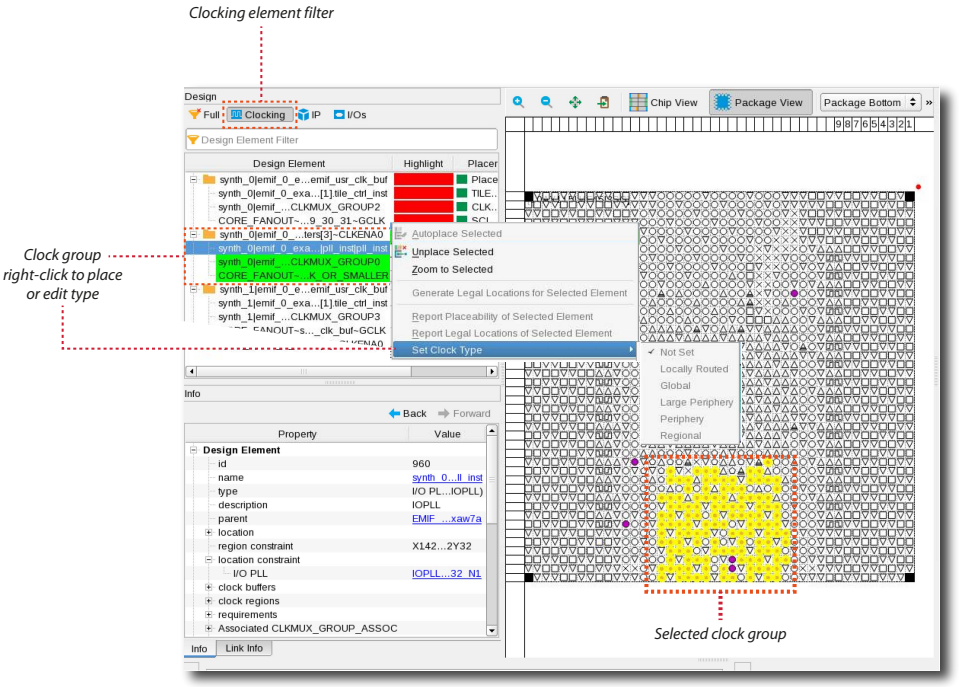
3.2.4.1 Plan Clock Networks

BluePrint allows you to visualize and plan clock networks. You can rapidly locate, highlight, place, and edit the type of clocking design elements in the **Plan** tab. BluePrint generates a Clocks report that details the signals using low-skew routing networks (clock networks) in the device.

To identify and place clocking elements in your design, click the **Clocking** filter in the **Plan** tab. You can refine the list further by entering search text in the **Design Element Filter**. BluePrint represents clock networks as groupings of the following clock network elements:

- Clock source
- Clock mux
- Clock region

Figure 22. Clocking Design Elements



You can place an entire clock group or individual clock elements using drag and drop, **Report Legal Locations of Selected Element**, or the **Autoplace Selected** commands. After placement, hover the mouse over the item in the **Design Element** list to highlight the placement. The placement of clock elements impacts the placement of dependent core and periphery elements.

You can edit the clock type for clocking design elements. The clock type impacts the placement of dependent core and periphery elements. Right-click any clock element to specify one of the following clock types:

- **Not Set**
- **Locally Routed**
- **Global**
- **Large Periphery**
- **Periphery**
- **Regional**

3.2.4.2 Saving & Loading Floorplans

You can save the state of your BluePrint floorplan for use in subsequent BluePrint sessions. BluePrint saves your plan in BluePrint Platform Designer Floorplan Format (`.plan`). You can load a `.plan` file in BluePrint to reopen the floorplan.

1. To save a BluePrint floorplan, click **File ➤ Save Floorplan** and specify a file name.
2. To load a BluePrint floorplan, click **File ➤ Load Floorplan** and browse for the `.plan` file.

Note: `.plan` files are for use only in BluePrint and are not for use directly in the Quartus Prime software. BluePrint generates an error if you attempt to load a `.plan` file that is not associated with the current BluePrint project.

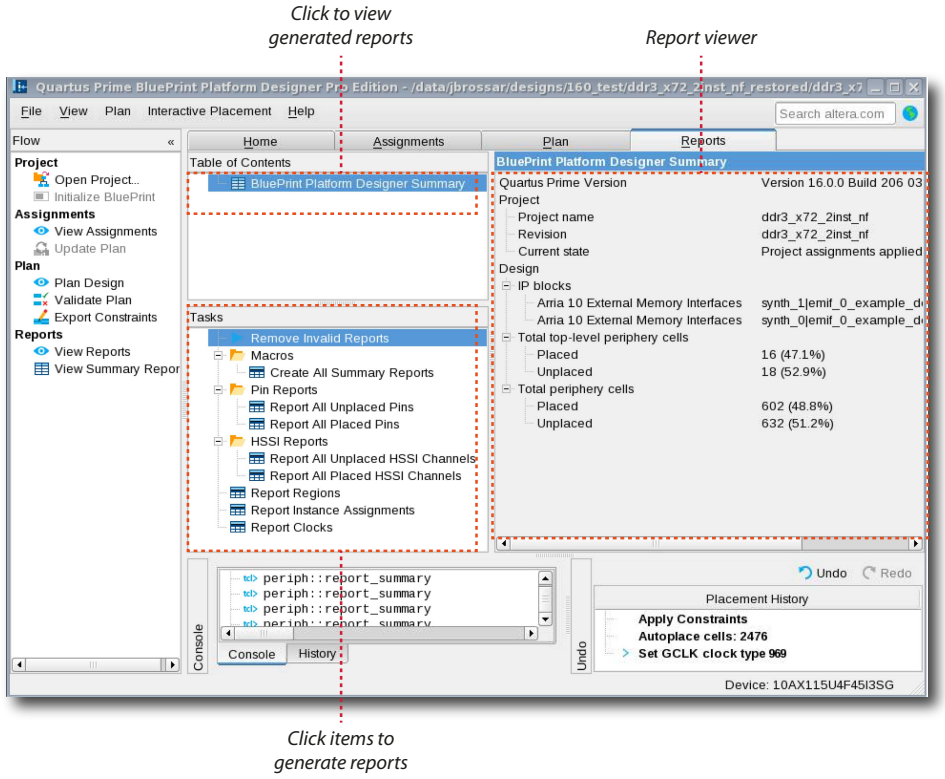
3.2.5 Step 5: Report Placement Data

Generate BluePrint placement and connectivity reports to help locate cells and make the best decisions about placement for the interfaces and elements in your design. Click **View Reports** on the **Flow** control to open the **Reports** tab from which you can generate a range of reports.

Click any report in the **Tasks** list to generate the report. The BluePrint reports are dynamic, allowing you to select elements in the report and place, unplace, or report detailed data about the selected element(s) or location(s).

Select any element in the **Plan** tab to **Report Placeability of Selected Element**, or **Report Legal Locations of Selected Element**. The generated reports appear on the **Reports** tab. This document describes each report in detail.

Figure 23. Reports Tab



Related Links

- Reports Tab Controls on page 57
  The BluePrint **Reports** tab contains the following **Task** pane controls to help you filter data and find entities and locations.
- BluePrint Reports on page 57

Use BluePrint reports to locate cells and assign suitable placement locations for specific interfaces and elements in your design.

3.2.6 Step 6: Validate and Export Plan Constraints

You must validate your BluePrint plan before exporting the plan constraints to your project as a generated Tcl script. Validation must confirm that the Fitter can place all remaining unplaced design elements. When you are satisfied with your BluePrint plan, follow these steps to validate and apply the BluePrint plan to your Quartus Prime project:

1. In the **Flow** control, click **Validate Plan**. The Fitter confirms placement of all remaining unplaced design elements. You must correct any errors before you can export constraints.
2. After validation, click **Export Constraints** to generate a Tcl script that applies the plan to your project. The output Tcl file contains instructions to apply the BluePrint plan to your Quartus Prime project. Close BluePrint and open the Quartus Prime software.
3. To apply the exported BluePrint plan constraints to your Quartus Prime project, click **Tools ➤ Tcl Scripts** and select the *<project name>*`.pdp_assignments.tcl` script file.
4. Click **Run**. The script runs, applying the BluePrint constraints to the project. Alternatively, you can run the script from the project directory:

```
quartus_sh –t <assignments_file>.tcl
```

5. In the Quartus Prime software, click **Start ➤ Start Analysis & Synthesis** to synthesize and apply the BluePrint plan in your project.

3.3 BluePrint User Interface Controls

The BluePrint user interface includes the following controls for planning your design platform.

Flow Controls on page 55
The **Flow** control panel provides immediate access to common BluePrint commands from anywhere within BluePrint.

Home Tab Controls on page 55
The BluePrint **Home** tab contains the following controls for opening projects in BluePrint.

Assignments Tab Controls on page 55
The **Assignments** tab contains controls for resolving potential conflicts with project assignments.

Plan Tab Controls on page 56
The **Plan** tab contains the following controls to help you place logic in the BluePrint plan.

Reports Tab Controls on page 57
The BluePrint **Reports** tab contains the following **Task** pane controls to help you filter data and find entities and locations.

3.3.1 Flow Controls

The **Flow** control panel provides immediate access to common BluePrint commands from anywhere within BluePrint. The **Flow** controls appear in order of a typical BluePrint planning flow.

Table 13. Flow Controls

Command	Description
Open Project	Allows you to select and open a Quartus Prime project in BluePrint.
Initialize BluePrint	Loads the synthesis netlist, starts the Fitter verification engine, and imports assignments from your Quartus Prime project.
View Assignments	Opens the **Assignments** tab, which allows you to review and reconcile any conflicting assignments that BluePrint imports from your project. Enable or disable specific project assignments to resolve any conflicts.
Update Plan	Applies the enabled project assignments to your BluePrint plan. You cannot perform periphery planning on the **Plan** tab until you update the plan.
Plan Design	Opens the **Plan** tab for placing logic in the BluePrint plan.
Export Constraints	Saves your BluePrint plan as a Tcl script file for subsequent application in your project. This command is available only after successfully running **Validate Plan**.
Validate plan	Verifies that all constraints in the BluePrint plan are compatible with placement of all remaining unplaced design elements. You can then directly locate and resolve the source of any reported validation errors. You must successfully validate the plan before running **Write Plan File**.
View Reports	Opens the **Reports** tab for filtering data and finding entities and locations.

3.3.2 Home Tab Controls

The BluePrint **Home** tab contains the following controls for opening projects in BluePrint.

Table 14. Home Tab Controls

Command	Description
Recent Projects	Provides quick access to your recently opened Quartus Prime projects. A named tile represents each project. Click the tile to display **Details** about the project. Double-click the tile to open the project in BluePrint.
Browse	Allows you to locate and open a Quartus Prime project in BluePrint. BluePrint requires your project's synthesized netlist for operation.
Details	Provides project and file details such as the file path, revision, and creation date of the Quartus Prime project. You can select a specific project revision.

3.3.3 Assignments Tab Controls

The **Assignments** tab contains controls for resolving potential conflicts with project assignments. Click **View Assignments** to display the **Assignments** tab.

You can enable or disable specific or classes of assignments until you resolve all potential conflicts. After you are satisfied with the status of all project assignments, click **Update Plan** to update your BluePrint plan with the enabled project assignments. BluePrint reports an error for any remaining assignment conflicts.

Table 15. Assignments Tab Controls

Command	Description
Filter field	Supports creation of wildcard expressions for assignment targets. **Enabled** and **Disabled** buttons filter only enabled or disabled assignments in the list.
Enable All Project Assignments	Enables all imported project assignments in your BluePrint plan.
Disable All Project Assignments	Disables all imported project assignments in the plan.
Clear	Clears any filter from the **Assignments** list.

3.3.4 Plan Tab Controls

The **Plan** tab contains the following controls to help you place logic in the BluePrint plan. Click **Plan Design** to display the **Plan** tab.

Placement or unplacement in the BluePrint plan does not apply to your Quartus Prime project until you add the generated BluePrint constraints script to your project.

Table 16. Plan Tab Controls

Command	Description
	Lists legal locations for placement.
Autoplace All	Attempts to place all unplaced design elements in legal locations in the BluePrint plan.
Autoplace Fixed	Attempts to place all unplaced design elements that have only one legal location into the BluePrint plan.
Unplace All	Unplaces all placed design elements in the BluePrint plan.
Right-click ➤ Auto-place selected element	Attempts to place the selected design element and all of its children in a legal location in the BluePrint plan.
Chip View	Displays the target device chip. Zoom in to display chip details.
Package View	Displays the target device package. Zoom in to display chip details.
Right-click ➤ Report Placeability of Selected Element	Displays detailed information on the **Reports** tab, showing legal locations in the BluePrint plan for the selected cell in order of suitability for fitting.
Copy Current View	Copies the current BluePrint plan to the clipboard for pasting into other files, such as word processing or presentation files.
Reset Plan	Unplaces all placed design elements and removes applied project assignments from the BluePrint plan. Resets all project assignments to the enabled state. You must subsequently run **Update Plan** prior to placing design elements. This command only applies to your BluePrint plan and does not impact your Quartus Prime project assignments until you apply the BluePrint script.
Load Floorplan	Allows you to select and load a BluePrint Platform Designer Floorplan Format (`.plan`) file. You can save BluePrint floorplan files in the format by clicking **Save Floorplan**.
Save Floorplan	Allows you to save your BluePrint floorplan as a `.plan` file.
Undo/Redo buttons	The **Undo** button reverts the last change made in the **Plan** tab. **Redo** re-implements the last undo. Use these commands to step forward and backward though your plan changes.

3.3.5 Reports Tab Controls

The BluePrint **Reports** tab contains the following **Task** pane controls to help you filter data and find entities and locations.

Table 17. Reports Tab Controls

Command	Description
Create all Summary Reports	Creates the following summary reports: • **BluePrint Platform Designer Summary** • **All Periphery Cells** • **Placed/Unplaced Periphery Cells** • **Periphery Location Types**.
Report All Placed/Unplaced Pins	Reports the name, parent (if any), and type of all placed (**Report All Placed Pins**) or unplaced (**Report All Unplaced Pins**) pins in the BluePrint plan, respectively. The Placed Pins report includes the placement location name. The Unplaced Pins report includes the number of potential placement locations. Right-click any cell to place, unplace, or report connectivity or location information.
Report All Placed/Unplaced HSSI Channels	Reports the name, parent (if any), and type of all placed (**Report All Placed HSSI Channels**) or unplaced (**Report All Unplaced HSSI Channels**) channels in the BluePrint plan, respectively. The Placed HSSI Channels report includes the placement location name. The Unplaced HSSI Channels report includes the number of potential placement locations. Right-click any cell to place, unplace, or report connectivity or location information.
Right-click ➤ Report Placed/ Unplaced Periphery Cells of Selected Type	Reports the name, parent (if any), and type of placed (**Report Placed Periphery Cells of Selected Type**) or unplaced (**Report Unplaced Periphery Cells of Selected Type**) cells matching the selected type. The placed cells report includes the placement location name. The unplaced cells report includes the number of potential placement locations. Right-click any cell to place, unplace, or report connectivity or location information.
Right-click ➤ Report Periphery Locations of Selected Type	Reports all locations in the device of the selected type, and whether the location supports merging.
Right-click ➤ Report Periphery Cell Connectivity	Reports the source port and type, destination port and type, of connections to the selected cell. Right-click any cell to report the individual cell connectivity.
Right-click ➤ Place/Unplace Cell	Places the cell in the selected location of the BluePrint plan. Similarly, you can right-click any cell and then click **Place Cell of Selected Type** or **Unplace Cell of Selected Type** to place or unplace multiple cells of the same type.
Right-click ➤ Report Cell Locations for Custom Placement	Reports the preferred legal locations for the selected cell in the BluePrint plan in the Legal Location report. Right-click to immediately place the cell in a location or report all periphery location of the selected type.
Remove Invalid Reports	Removes outdated BluePrint reports that you invalidate by changes to the BluePrint plan.
Report Instance Assignments	Shows all imported project assignments in the BluePrint plan. You can delete these assignments from the plan.

3.4 BluePrint Reports

Use BluePrint reports to locate cells and assign suitable placement locations for specific interfaces and elements in your design. BluePrint reports provide detailed, actionable feedback to help you quickly implement the best plan for your design. You can access placement and further reporting functions directly from BluePrint reports. BluePrint generates the following reports that provide detailed planning information:

Report Summary on page 58

Click **Create all Summary Reports** on the **Reports** tab to generate summary reports about periphery cells in the BluePrint plan.

Report Pins on page 59
Generate reports about I/O pins in the design.

Report HSSI Channels on page 60
Generate reports about HSSI channels in the BluePrint plan.

Report Clocks on page 61
Generate reports showing clock networks in the plan.

Report Periphery Locations on page 61
Generate reports that show the status of periphery cells in the BluePrint plan.

Report Cell Connectivity on page 62
Generate reports showing the connections between all cells in the BluePrint plan.

Report Instance Assignments on page 62
Click **Report Instance Assignments** to show all imported project assignments in the BluePrint plan.

3.4.1 Report Summary

Click **Create all Summary Reports** on the **Reports** tab to generate summary reports about periphery cells in the BluePrint plan.
Right-click any cell type to report placed, unplaced, connectivity, or location information.

Figure 24. Summary Reports

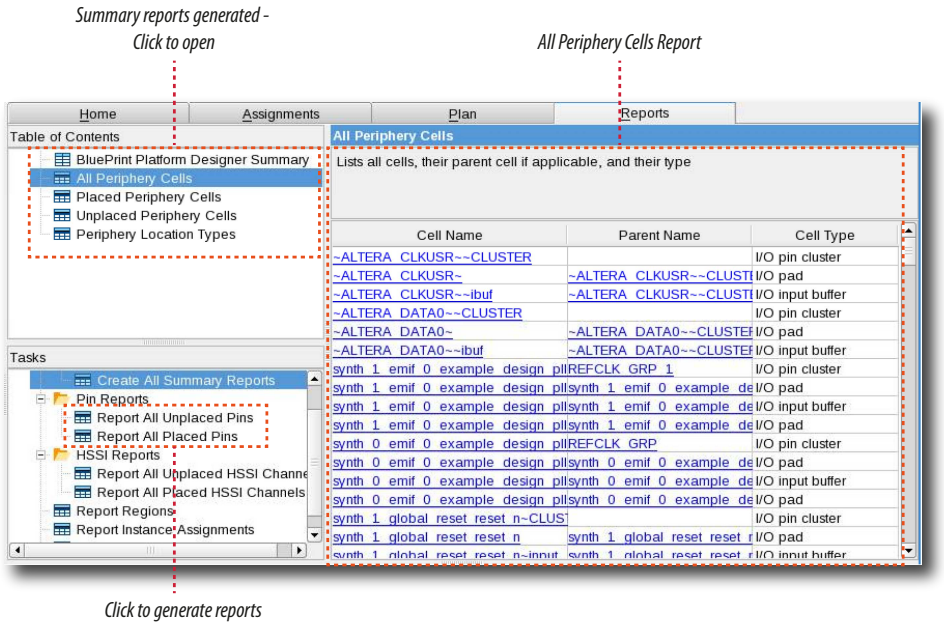


Table 18. Report Summary

Command	Description
Create all Summary Reports	Creates the following summary reports:

Command	Description
	• **BluePrint Platform Designer Summary**—reports software version and total number of periphery and top-level periphery cells. • **All Periphery Cells**— reports the name, parent, and type of all periphery cells in the design. • **Placed/Unplaced Periphery Cells**—reports the name, parent, and type of all placed and unplaced periphery cells in the BluePrint plan. • **Periphery Location Types**—reports the number of each type of periphery location available in the target device and the number required by your design.

3.4.2 Report Pins

Generate reports about I/O pins in the design. Right-click any cell type to place, unplace, or report connectivity or location information.

Table 19. Report Pin Commands

Command	Description
Report All Placed Pins	Generates the Placed Pins report. This report lists the name, parent, type, and location of all placed pins in the BluePrint plan.
Report All Unplaced Pins	Generates the Unplaced Pins report. This report lists the name, parent, type, and the number of potential placements for all unplaced pins in the BluePrint plan.

Figure 25. Placed Pins Report

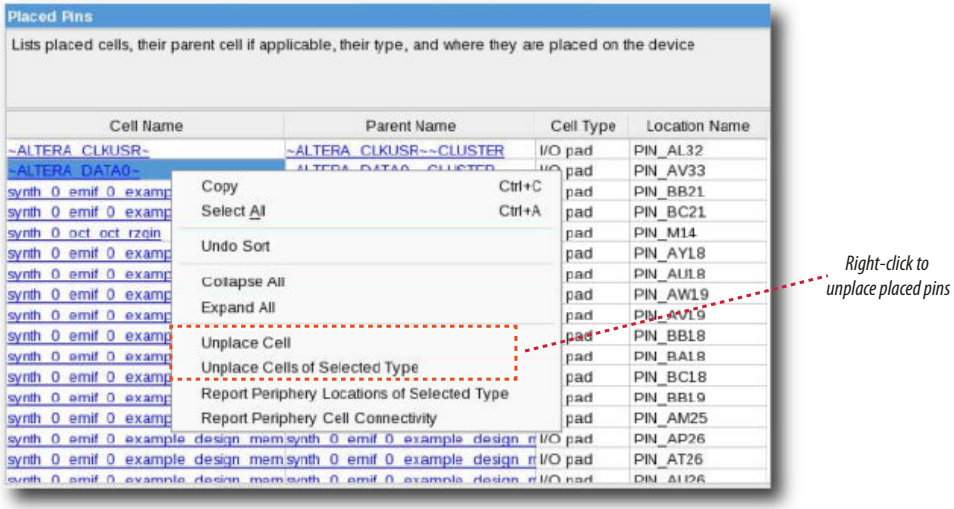
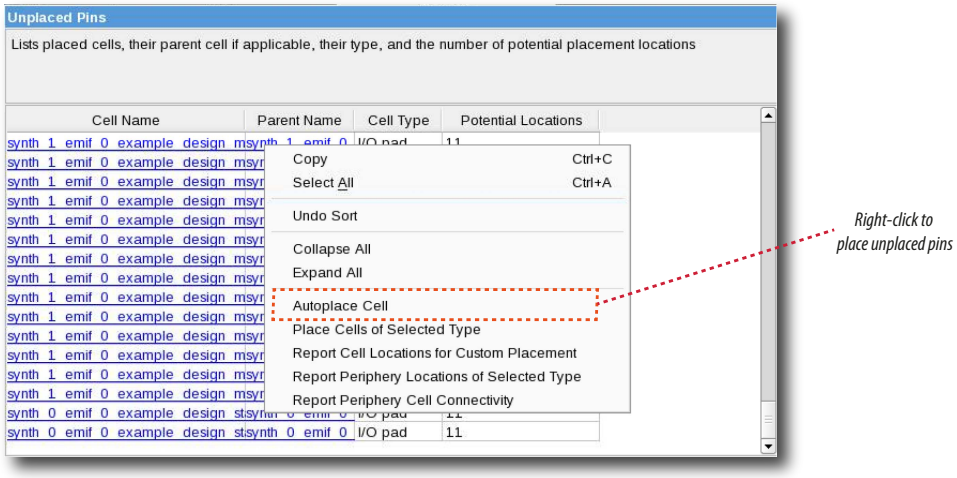


Figure 26. Unplaced Pins Report



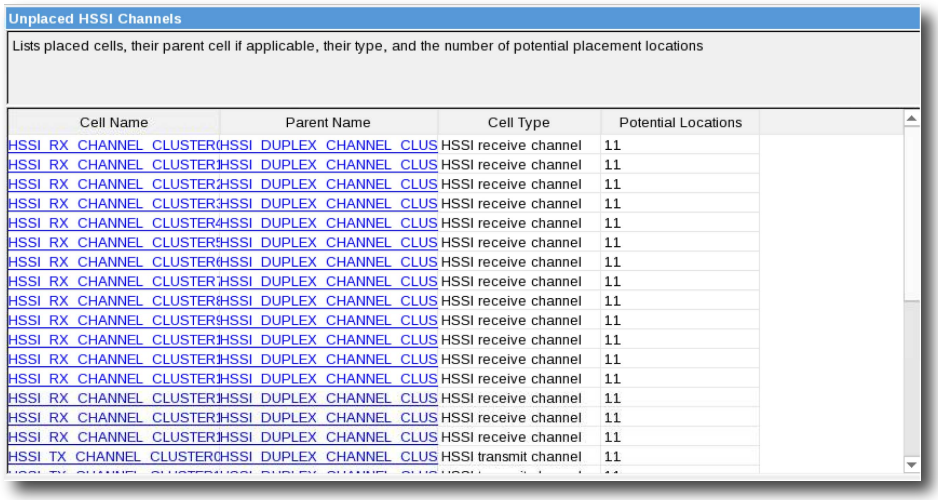
3.4.3 Report HSSI Channels

Generate reports about HSSI channels in the BluePrint plan. Right-click any cell type to place, unplace, or report connectivity or location information.

Table 20. Report Channel Commands

Command	Description
Report All Placed HSSI Channels	Generates the Placed HSSI Channels report. This report lists the name, parent, type, and location of all placed HSSI RX/TX channels in the BluePrint plan.
Report All Unplaced HSSI Channels	Generates the Unplaced HSSI Channels report. This report lists the name, parent, type, and location of all unplaced HSSI RX/TX channels in the BluePrint plan.

Figure 27. Unplaced HSSI Channels Report



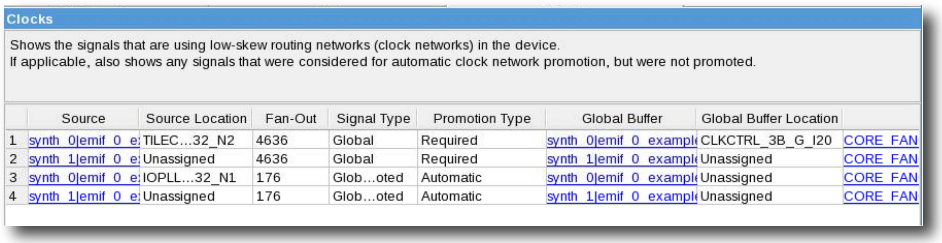
3.4.4 Report Clocks

Generate reports showing clock networks in the plan. Use this report to analyze clock network scenarios and ensure that specific device regions are fed by high fan-out signals.

Table 21. Report Clocks Commands

Command	Description
Report Clocks	Generates the Global and other Fast Signals report.

Figure 28. Clocks Report



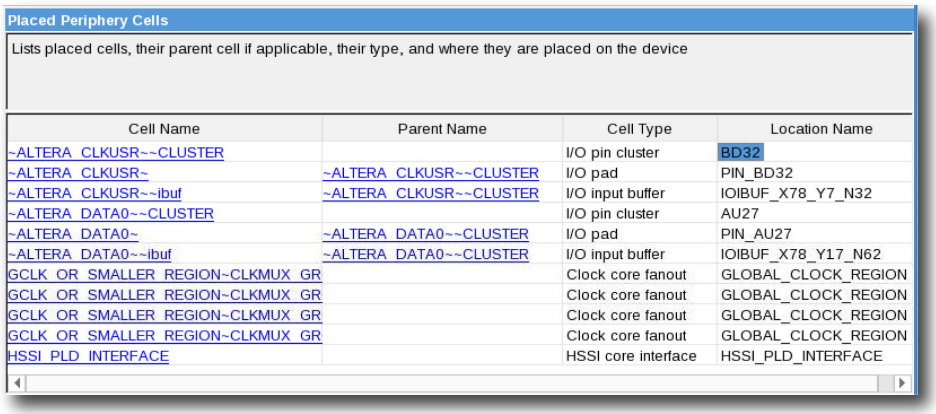
3.4.5 Report Periphery Locations

Generate reports that show the status of periphery cells in the BluePrint plan.

Table 22. Report Periphery Locations Commands

Command	Description
Right-click ➤ Report Placed Periphery Cells of Selected Type	Accessible from the **All Periphery Cells** report. This command reports the name, parent (if any), type, and location of the selected placed periphery cells matching the selected type. Right-click any cell to place, unplace, or report connectivity or location information.
Right-click ➤ Report Unplaced Periphery Cells of Selected Type	Accessible from the **All Periphery Cells** report. This command reports the name, parent (if any), type, and number of suitable locations for the selected unplaced periphery cells matching the selected type. Right-click any cell to place, unplace, or report connectivity or location information.
Right-click ➤ Report Periphery Locations of Selected Type	Reports all locations in the device of the selected type, and whether the location supports merging.

Figure 29. Periphery Locations Report



Placed Periphery Cells

Lists placed cells, their parent cell if applicable, their type, and where they are placed on the device

Cell Name	Parent Name	Cell Type	Location Name
~ALTERA_CLKUSR~~CLUSTER		I/O pin cluster	BD32
~ALTERA_CLKUSR~	~ALTERA_CLKUSR~~CLUSTER	I/O pad	PIN_BD32
~ALTERA_CLKUSR~~ibuf	~ALTERA_CLKUSR~~CLUSTER	I/O input buffer	IOIBUF_X78_Y7_N32
~ALTERA_DATA0~~CLUSTER		I/O pin cluster	AU27
~ALTERA_DATA0~	~ALTERA_DATA0~~CLUSTER	I/O pad	PIN_AU27
~ALTERA_DATA0~~ibuf	~ALTERA_DATA0~~CLUSTER	I/O input buffer	IOIBUF_X78_Y17_N62
GCLK_OR_SMALLER_REGION~CLKMUX_GR		Clock core fanout	GLOBAL_CLOCK_REGION
GCLK_OR_SMALLER_REGION~CLKMUX_GR		Clock core fanout	GLOBAL_CLOCK_REGION
GCLK_OR_SMALLER_REGION~CLKMUX_GR		Clock core fanout	GLOBAL_CLOCK_REGION
GCLK_OR_SMALLER_REGION~CLKMUX_GR		Clock core fanout	GLOBAL_CLOCK_REGION
HSSI_PLD_INTERFACE		HSSI core interface	HSSI_PLD_INTERFACE

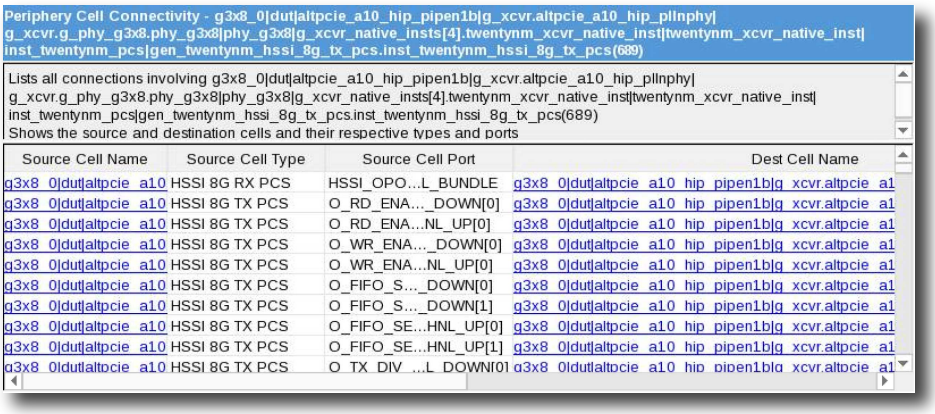
3.4.6 Report Cell Connectivity

Generate reports showing the connections between all cells in the BluePrint plan.

Table 23. Report Cell Connectivity Command

Command	Description
Right-click ➤ Report Periphery Cell Connectivity	Right-click any **Cell Name** in the reports to **Report Periphery Cell Connectivity**. The report lists the source and destination ports and type of connections to the selected cell. Right-click any cell to report all connections to the cell.

Figure 30. Cell Connectivity Report



Periphery Cell Connectivity - g3x8_0|dut|altpcie_a10_hip_pipen1b|g_xcvr.altpcie_a10_hip_pllnphy|g_xcvr.g_phy_g3x8.phy_g3x8|phy_g3x8|g_xcvr_native_insts[4].twentynm_xcvr_native_inst|twentynm_xcvr_native_inst|inst_twentynm_pcs|gen_twentynm_hssi_8g_tx_pcs.inst_twentynm_hssi_8g_tx_pcs(689)

Lists all connections involving g3x8_0|dut|altpcie_a10_hip_pipen1b|g_xcvr.altpcie_a10_hip_pllnphy|g_xcvr.g_phy_g3x8.phy_g3x8|phy_g3x8|g_xcvr_native_insts[4].twentynm_xcvr_native_inst|twentynm_xcvr_native_inst|inst_twentynm_pcs|gen_twentynm_hssi_8g_tx_pcs.inst_twentynm_hssi_8g_tx_pcs(689)
Shows the source and destination cells and their respective types and ports

Source Cell Name	Source Cell Type	Source Cell Port	Dest Cell Name
g3x8_0|dut|altpcie_a10	HSSI 8G RX PCS	HSSI_OPO...L_BUNDLE	g3x8_0|dut|altpcie_a10_hip_pipen1b|g_xcvr.altpcie_a1
g3x8_0|dut|altpcie_a10	HSSI 8G TX PCS	O_RD_ENA..._DOWN[0]	g3x8_0|dut|altpcie_a10_hip_pipen1b|g_xcvr.altpcie_a1
g3x8_0|dut|altpcie_a10	HSSI 8G TX PCS	O_RD_ENA...NL_UP[0]	g3x8_0|dut|altpcie_a10_hip_pipen1b|g_xcvr.altpcie_a1
g3x8_0|dut|altpcie_a10	HSSI 8G TX PCS	O_WR_ENA..._DOWN[0]	g3x8_0|dut|altpcie_a10_hip_pipen1b|g_xcvr.altpcie_a1
g3x8_0|dut|altpcie_a10	HSSI 8G TX PCS	O_WR_ENA...NL_UP[0]	g3x8_0|dut|altpcie_a10_hip_pipen1b|g_xcvr.altpcie_a1
g3x8_0|dut|altpcie_a10	HSSI 8G TX PCS	O_FIFO_S..._DOWN[0]	g3x8_0|dut|altpcie_a10_hip_pipen1b|g_xcvr.altpcie_a1
g3x8_0|dut|altpcie_a10	HSSI 8G TX PCS	O_FIFO_S..._DOWN[1]	g3x8_0|dut|altpcie_a10_hip_pipen1b|g_xcvr.altpcie_a1
g3x8_0|dut|altpcie_a10	HSSI 8G TX PCS	O_FIFO_SE...HNL_UP[0]	g3x8_0|dut|altpcie_a10_hip_pipen1b|g_xcvr.altpcie_a1
g3x8_0|dut|altpcie_a10	HSSI 8G TX PCS	O_FIFO_SE...HNL_UP[1]	g3x8_0|dut|altpcie_a10_hip_pipen1b|g_xcvr.altpcie_a1
g3x8_0|dut|altpcie_a10	HSSI 8G TX PCS	O_TX_DIV_...L_DOWN[0]	g3x8_0|dut|altpcie_a10_hip_pipen1b|g_xcvr.altpcie_a1

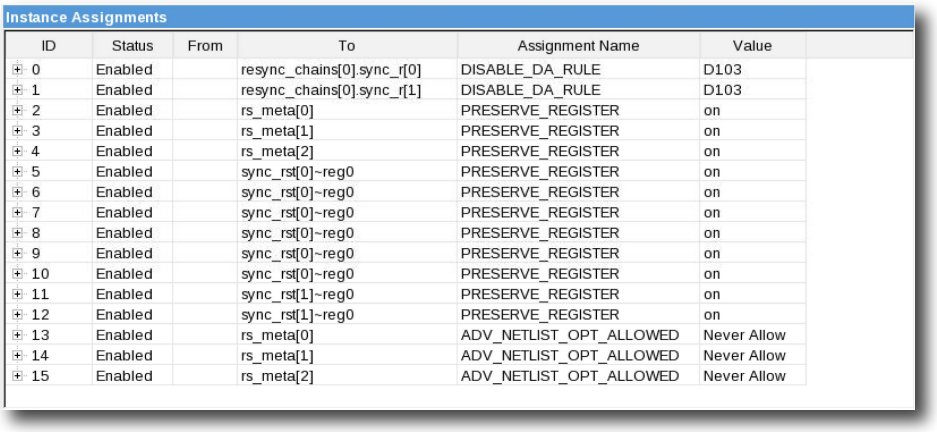
3.4.7 Report Instance Assignments

Click **Report Instance Assignments** to show all imported project assignments in the BluePrint plan. You can delete these assignments from the plan.

Table 24. Report Instance Assignments Command

Command	Description
Report Instance Assignments	Reports all enabled instance assignments in your design. Right-click any cell to delete the assignment or to delete all assignments of the same type.

Figure 31. Instance Assignments Report



3.5 Document Revision History

This document has the following revision history.

Date	Version	Changes
2016.10.31	16.1.0	• Implemented Intel rebranding.
2016.05.03	16.0.0	• Added Plan Clock Networks topic. • Added Saving and Loading Floorplans topic. • Added Undo/Redo command descriptions. • Added Flow control description. • Added note about panning feature. • Updated all screenshots for latest GUI.
2015.11.02	15.1.0	• Changed instances of *Quartus II* to *Quartus Prime*. • Integration of content into Quartus Prime Handbook. • Added descriptions of new dynamic reports. • Added Package View description. • Added GUI controls reference.
2015.05.04	15.0.0	Second beta release of document on Molson. Added information about the following subjects: • Overview information • Reset Plan command • Legal Assignments list and prompt • Tcl console
2014.12.15	14.1.	First beta release of document on Molson.

Related Links

Altera Documentation Archive

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

4 Command Line Scripting

FPGA design software that easily integrates into your design flow saves time and improves productivity. The Intel Quartus Prime software provides you with a command-line executable for each step of the FPGA design flow to make the design process customizable and flexible.

The command-line executables are completely interchangeable with the Quartus Prime GUI, allowing you to use the exact combination of tools that best suits your needs.

4.1 Benefits of Command-Line Executables

Quartus Prime command-line executables give you precise control over each step of the design flow, reduce memory requirements, and improve performance.

You can group Quartus Prime executable files into a script, batch file, or a makefile to automate design flows. These scripting capabilities facilitate the integration of Quartus Prime software and other EDA synthesis, simulation, and verification software. Automatic design flows can perform on multiple computers simultaneously and easily archive and restore projects.

Command-line executables add flexibility without sacrificing the ease-of-use of the Quartus Prime GUI. You can use the Quartus Prime GUI and command-line executables at different stages in the design flow. For example, you might use the Quartus Prime GUI to edit the floorplan for the design, use the command-line executables to perform place-and-route, and return to the Quartus Prime GUI to perform debugging.

Command-line executables reduce the amount of memory required during each step in the design flow. Since each executable targets only one step in the design flow, the executables themselves are relatively compact, both in file size and the amount of memory used during processing. This memory usage reduction improves performance, and is particularly beneficial in design environments where heavy usage of computing resources results in reduced memory availability.

Related Links

About Command-Line Executables
in Quartus Prime Help

4.2 Command-Line Scripting Help

Help for command-line executables is available through different methods. You can access help built into the executables with command-line options. You can use the Quartus Prime Command-Line and Tcl API Help browser for an easy graphical view of the help information.

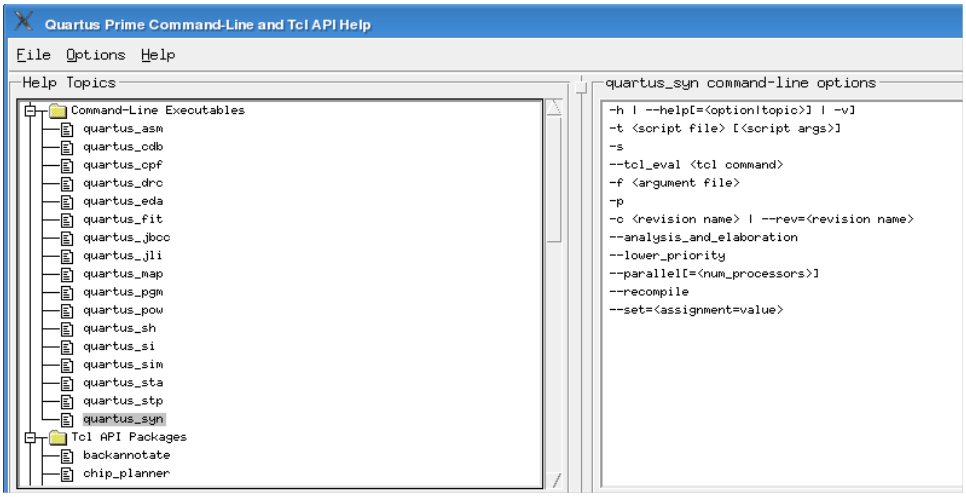
To use the Quartus Prime Command-Line and Tcl API Help browser, type the following command:

```
quartus_sh --qhelp
```

This command starts the Quartus Prime Command-Line and Tcl API Help browser, a viewer for information about the Quartus Prime Command-Line executables and Tcl API.

Use the `-h` option with any of the Quartus Prime Command-Line executables to get a description and list of supported options. Use the `--help=`<*option name*> option for detailed information about each option.

Figure 32. Quartus Prime Command-Line and Tcl API Help Browser



4.3 Project Settings with Command-Line Options

The Quartus Prime software command-line executables accept arguments to set project variables and access common settings.

To make assignments to an individual entity you can use the Quartus Prime Tcl scripting API. On existing projects, you can also open the project in the Quartus Prime GUI, change the assignment, and close the project. The changed assignment is updated in the `.qsf`. Any command-line executables that are run after this update use the updated assignment.

Related Links

- Tcl Scripting on page 72
- Quartus Prime Settings File (.qsf) Definition
  in Quartus Prime Help
- Quartus Prime Pro Edition Settings File Reference Manual
  For information about all settings and constraints in the Quartus Prime software.

4.3.1 Option Precedence

Project assignments follow a set of precedence rules. Assignments for a project can exist in three places:

- Quartus Prime Settings File (`.qsf`)
- The compiler database
- Command-line options

The `.qsf` file contains all the project-wide and entity-level assignments and settings for the current revision for the project. The compiler database contains the result of the last compilation in the `/db` directory, and reflects the assignments at the moment when the project was compiled. Updated assignments first appear in the compiler database and later in the `.qsf` file.

Command-line options override any conflicting assignments in the `.qsf` file or the compiler database files. To specify whether the `.qsf` or compiler database files take precedence for any assignments not specified in the command-line, use the option `--read_settings_files`.

Table 25. Precedence for Reading Assignments

Option Specified	Precedence for Reading Assignments
`--read_settings_files = on` (Default)	1. Command-line options 2. The `.qsf` for the project 3. Project database (db directory, if it exists) 4. Quartus Prime software defaults
`--read_settings_files = off`	1. Command-line options 2. Project database (db directory, if it exists) 3. Quartus Prime software defaults

The `--write_settings_files` command-line option lists the locations to which assignments are written..

Table 26. Location for Writing Assignments

Option Specified	Location for Writing Assignments
`--write_settings_files = on` (Default)	`.qsf` file and compiler database
`--write_settings_files = off`	Compiler database

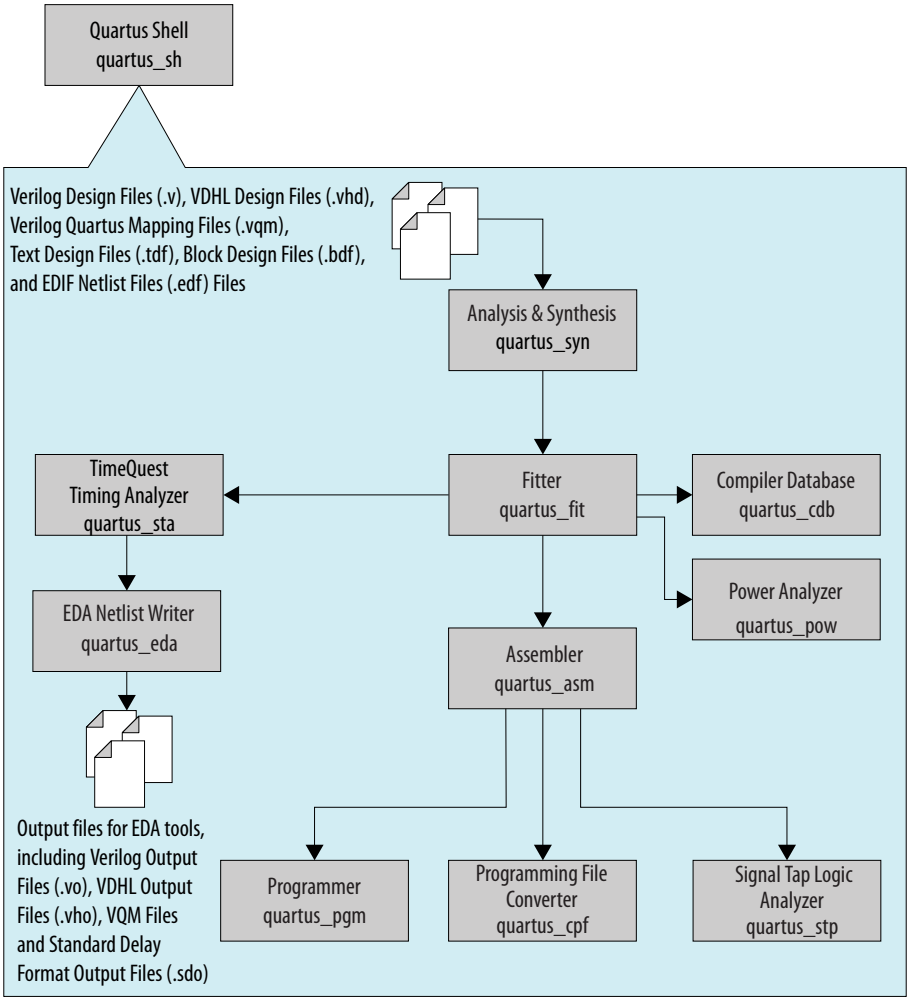
Any assignment not specified as a command-line option or found in the `.qsf` file or compiler database file is set to its default value.

Use the options `--read_settings_files=off` and `--write_settings_files=off` (where appropriate) to optimize the way that the Quartus Prime software reads and updates settings files.

4.4 Compilation with quartus_sh --flow

The figure shows a typical Quartus Prime FPGA design flow using command-line executables.

Figure 33. Typical Design Flow



Use the `quartus_sh` executable with the `--flow` option to perform a complete compilation flow with a single command. The `--flow` option supports the smart recompile feature and efficiently sets command-line arguments for each executable in the flow.

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

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

Tip: For information about specialized flows, type `quartus_sh --help=flow` at a command prompt.

4.5 Text-Based Report Files

Each command-line executable creates a text report file when it is run. These files report success or failure, and contain information about the processing performed by the executable.

Report file names contain the revision name and the short-form name of the executable that generated the report file, in the format `<revision>.<executable>.rpt`. For example, using the `quartus_fit` executable to place and route a project with the revision name **design_top** generates a report file named `design_top.fit.rpt`. Similarly, using the `quartus_sta` executable to perform timing analysis on a project with the revision name **fir_filter** generates a report file named `fir_filter.sta.rpt`.

As an alternative to parsing text-based report files, you can use the `::quartus::report` Tcl package.

Related Links

- Text-Format Report File (.rpt) Definition
  in Quartus Prime Help
- ::quartus::report
  in Quartus Prime Help

4.6 Using Command-Line Executables in Scripts

You can use command-line executables in scripts that control other software, in addition to Quartus Prime software. For example, if your design flow uses third-party synthesis or simulation software, and you can run this other software at the command prompt, you can group those commands with Quartus Prime executables in a single script.

To set up a new project and apply individual constraints, such as pin location assignments and timing requirements, you must use a Tcl script or the Quartus Prime GUI.

Command-line executables are very useful for working with existing projects, for making common global settings, and for performing common operations. For more flexibility in a flow, use a Tcl script. Additionally, Tcl scripts simplify passing data between different stages of the design flow.

For example, you can create a UNIX shell script to run a third-party synthesis software, place-and-route the design in the Quartus Prime software, and generate output netlists for other simulation software.

4.7 The QFlow Script

A Tcl/Tk-based graphical interface called QFlow is included with the command-line executables. You can use the QFlow interface to open projects, launch some of the command-line executables, view report files, and make some global project assignments.

The QFlow interface can run the following command-line executables:

- `quartus_syn` (Analysis and Synthesis)
- `quartus_fit` (Fitter)
- `quartus_sta` (TimeQuest timing analyzer)
- `quartus_asm` (Assembler)
- `quartus_eda` (EDA Netlist Writer)

To view floorplans or perform other GUI-intensive tasks, launch the Quartus Prime software.

Start QFlow by typing the following command at a command prompt:

```
quartus_sh -g
```

Tip: The QFlow script is located in the *<Quartus Prime directory>*`/common/tcl/apps/qflow/` directory.

4.8 Document Revision History

Table 27. Document Revision History

Date	Version	Changes
2017.05.08	17.0.0	• Reorganized content on topics: Benefits of Command-Line Executables and Project Settings with Command-Line Options. • Removed mentions to unsupported executables and options. • Removed topics: Introductory Example and Common Scripting Examples
2016.10.31	16.1.0	• Implemented Intel rebranding.
2015.11.02	15.1.0	Changed instances of *Quartus II* to *Quartus Prime*.
2015.05.04	15.0.0	Remove descriptions of makefile support that was removed from software in 14.0.
December 2014	14.1.0	Updated DSE II commands.
June 2014	14.0.0	Updated formatting.
November 2013	13.1.0	Removed information about -silnet `qmegawiz` command
June 2012	12.0.0	Removed survey link.
November 2011	11.0.1	Template update.
May 2011	11.0.0	Corrected quartus_qpf example usage. Updated examples.
December 2010	10.1.0	Template update. Added section on using a script to regenerate megafunction variations. Removed references to the Classic Timing Analyzer (quartus_tan). Removed Qflow illustration.
July 2010	10.0.0	Updated script examples to use quartus_sta instead of quartus_tan, and other minor updates throughout document.
November 2009	9.1.0	Updated Table 2–1 to add quartus_jli and quartus_jbcc executables and descriptions, and other minor updates throughout document.

continued...

Date	Version	Changes
March 2009	9.0.0	No change to content.
November 2008	8.1.0	Added the following sections: • "The MegaWizard Plug-In Manager" on page 2–11 "Command-Line Support" on page 2–12 "Module and Wizard Names" on page 2–13 "Ports and Parameters" on page 2–14 "Invalid Configurations" on page 2–15 "Strategies to Determine Port and Parameter Values" on page 2–15 "Optional Files" on page 2–15 "Parameter File" on page 2–16 "Working Directory" on page 2–17 "Variation File Name" on page 2–17 • "Create a Compressed Configuration File" on page 2–21 • Updated "Option Precedence" on page 2–5 to clarify how to control precedence • Corrected Example 2–5 on page 2–8 • Changed Example 2–1, Example 2–2, Example 2–4, and Example 2–7 to use the EP1C12F256C6 device • Minor editorial updates • Updated entire chapter using 8½" × 11" chapter template
May 2008	8.0.0	• Updated "Referenced Documents" on page 2–20. • Updated references in document.

Related Links

Altera Documentation Archive

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

5 Tcl Scripting

5.1 Tcl Scripting

You can use Tcl scripts to control the Intel® Quartus Prime software and to perform a wide range of functions, such as compiling a design or scripting common tasks.

For example, use Tcl scripts to perform the following tasks:

- Manage a Quartus Prime project
- Make assignments
- Define design constraints
- Make device assignments
- Compile your design
- Perform timing analysis
- Access reports

Tcl scripts also facilitate project or assignment migration. For example, when designing in different projects with the same prototype or development board, you can write a script to automate reassignment of pin locations in each new project. The Quartus Prime software can also generate a Tcl script based on all the current assignments in the project, which aids in switching assignments to another project.

The Quartus Prime software Tcl commands follow the EDA industry Tcl application programming interface (API) standards for command-line options. This simplifies learning and using Tcl commands. If you encounter an error with a command argument, the Tcl interpreter includes help information showing correct usage.

This chapter includes sample Tcl scripts for automating tasks in the Quartus Prime software. You can modify these example scripts for use with your own designs. You can find more Tcl scripts in the Design Examples section of the Support area on the Altera website.

Related Links

Design Examples

5.2 Tool Command Language

Tcl (pronounced “tickle”) stands for Tool Command Language, and is the industry-standard scripting language. Tcl supports control structures, variables, network socket access, and APIs.

With Tcl, you can work seamlessly across most development platforms. Synopsys, Mentor Graphics, and Intel software products support the Tcl language.

By combining Tcl commands and Quartus Prime API functions, you can create your own procedures and automate your design flow. Run Quartus Prime software in batch mode, or execute individual Tcl commands interactively in the Quartus Prime Tcl shell.

Quartus Prime software supports Tcl/Tk version 8.5, supplied by the Tcl DeveloperXchange.

Related Links

- External References on page 94
  For more information about Tcl, refer to the following sources:
- Tcl Scripting Basics on page 88
  The core Tcl commands support variables, control structures, and procedures. Additionally, there are commands for accessing the file system and network sockets, and running other programs. You can create platform-independent graphical interfaces with the Tk widget set.
- tcl.activestate.com

5.3 Quartus Prime Tcl Packages

The Quartus Prime software groups Tcl commands into packages by function.

Table 28. Quartus Prime Tcl Packages

Package Name	Package Description
chip_planner	Identify and modify resource usage and routing with the Chip Editor
design	Manipulate project databases, including the assignments database, to enable the creation of instance assignments without modifying the `.qsf` file
device	Get device and family information from the device database
external_memif_toolkit	Interact with external memory interfaces and debug components
fif	Contains the set of Tcl functions for using the Fault Injection File (FIF) Driver
flow	Compile a project, run command-line executables, and other common flows
insystem_memory_edit	Read and edit memory contents in Intel devices
insystem_source_probe	Interact with the In-System Sources and Probes tool in an Intel device
interactive_synthesis	
iptclgen	Generate memory IP
jtag	Control the JTAG chain
logic_analyzer_interface	Query and modify the Logic Analyzer Interface output pin state
misc	Perform miscellaneous tasks such as enabling natural bus naming, package loading, and message posting
names	
periph	Interact with the BluePrint plans
project	Create and manage projects and revisions, make any project assignments including timing assignments
report	Get information from report tables, create custom reports
rtl	Traverse and query the RTL netlist of your design

continued...

Package Name	Package Description
rtm	
sdc	Specify constraints and exceptions to the TimeQuest Timing Analyzer
sdc_ext	Intel-specific SDC commands
simulator	Configure and perform simulations
sta	Contain the set of Tcl functions for obtaining advanced information from the TimeQuest Timing Analyzer
stp	Run the Signal Tap Logic Analyzer
synthesis_report	Contain the set of Tcl functions for the Dynamic Synthesis Report tool
tdc	Obtain information from the TimeQuest Timing Analyzer

To keep memory requirements as low as possible, only the minimum number of packages load automatically with each Quartus Prime executable. To run commands from other packages, load those packages beforehand.

Run your scripts with executables that include the packages you use in the scripts. For example, to use commands in the `sdc_ext` package, you must use the `quartus_sta` executable because `quartus_sta` is the only executable with support for the `sdc_ext` package.

The following command prints lists of the packages loaded or available to load for an executable, to the console:

`<executable name> --tcl_eval help`

For example, type the following command to list the packages loaded or available to load by the `quartus_fit` executable:

```
quartus_fit --tcl_eval help
```

5.3.1 Loading Packages

To load a Quartus Prime Tcl package, use the `load_package` command as follows:

```
load_package [-version <version number>] <package name>
```

This command is similar to `package require`, but it allows to alternate between different versions of a Quartus Prime Tcl package.

Related Links

Command Line Scripting on page 65
FPGA design software that easily integrates into your design flow saves time and improves productivity. The Intel Quartus Prime software provides you with a command-line executable for each step of the FPGA design flow to make the design process customizable and flexible.

5.4 Quartus Prime Tcl API Help

You can access the Quartus Prime Tcl API Help by typing the following at a system command prompt:

```
quartus_sh --qhelp
```

This command runs the Quartus Prime Command-Line and Tcl API help browser, which documents all commands and options in the Quartus Prime Tcl API.

Quartus Prime Tcl help allows easy access to information about the Quartus Prime Tcl commands. To access the help information, type `help` at a Tcl prompt.

Tcl Help Output

Table 29. Help Options Available in the Quartus Prime Tcl Environment

Help Command	Description
`help`	To view a list of available Quartus Prime Tcl packages, loaded and not loaded.
`help -tcl`	To view a list of commands used to load Tcl packages and access command-line help.

`help -pkg <package_name> [-version <version number>]`	To view help for a specified Quartus Prime package that includes the list of available Tcl commands. For convenience, you can omit the `::quartus::` package prefix, and type `help -pkg <package name>`.
	If you do not specify the `-version` option, help for the currently loaded package is displayed by default. If the package for which you want help is not loaded, help for the latest version of the package is displayed by default.
	Examples:
	`help -pkg ::quartus::project`
	`help -pkg project help -pkg project -version 1.0`

`<command_name> -h`
or
`<command_name> -help`
	To view short help for a Quartus Prime Tcl command for which the package is loaded.
	Examples:
	`project_open -h`
	`project_open -help`

`package require ::quartus::<package name> [<version>]`	To load a Quartus Prime Tcl package with the specified version. If <*version*> is not specified, the latest version of the package is loaded by default.
	Example:
	`package require ::quartus::project 1.0`
	This command is similar to the `load_package` command.
	The advantage of the `load_package` command is that you can alternate freely between different versions of the same package.
	Type `load_package <package name> [-version <version number>]`to load a Quartus Prime Tcl package with the specified version. If the `-version` option is not specified, the latest version of the package is loaded by default.
	Example:
	`load_package ::quartus::project -version 1.0`

`help -cmd <command_name> [-version <version>]`
or
`<command_name> -long_help`
	To view complete help text for a Quartus Prime Tcl command.
	If you do not specify the `-version` option, help for the command in the currently loaded package version is displayed by default.
	If the package version for which you want help is not loaded, help for the latest version of the package is displayed by default.
	Examples:
	`project_open -long_help`
	`help -cmd project_open`

continued...

Help Command	Description
	`help -cmd project_open -version 1.0`
`help -examples`	To view examples of Quartus Prime Tcl usage.
`help -quartus`	To view help on the predefined global Tcl array that contains project information and information about the Quartus Prime executable that is currently running.
`quartus_sh --qhelp`	To launch the Tk viewer for Quartus Prime command-line help and display help for the command-line executables and Tcl API packages.
`help -timequestinfo`	To view help on the predefined global "TimeQuestInfo" Tcl array that contains delay model information and speed grade information of a TimeQuest design that is currently running.

The Tcl API help is also available in Quartus Prime online help. Search for the command or package name to find details about that command or package.

5.4.1 Command-Line Options

You can use any of the following command line options with executables that support Tcl:

Table 30. Command-Line Options Supporting Tcl Scripting

Command-Line Option	Description
`--script=<script file> [<script args>]`	Run the specified Tcl script with optional arguments.
`-t <script file> [<script args>]`	Run the specified Tcl script with optional arguments. The `-t` option is the short form of the `--script` option.
`--shell`	Open the executable in the interactive Tcl shell mode.
`-s`	Open the executable in the interactive Tcl shell mode. The `-s` option is the short form of the `--shell` option.
`--tcl_eval <tcl command>`	Evaluate the remaining command-line arguments as Tcl commands. For example, the following command displays help for the project package: `quartus_sh --tcl_eval help -pkg project`

5.4.1.1 Run a Tcl Script

Running an executable with the `-t` option runs the specified Tcl script. You can also specify arguments to the script. Access the arguments through the `argv` variable, or use a package such as `cmdline`, which supports arguments of the following form:

```
-<argument name> <argument value>
```

The `cmdline` package is included in the *<Quartus Prime directory>*`/common/tcl/packages` directory.

For example, to run a script called `myscript.tcl` with one argument, `Stratix`, type the following command at a system command prompt:

```
quartus_sh -t myscript.tcl Stratix
```

5.4.1.2 Interactive Shell Mode

Running an executable with the `-s` option starts an interactive Tcl shell. For example, to open the Quartus Prime TimeQuest Timing Analyzer executable in interactive shell mode, type:

```
quartus_sta -s
```

Commands you type in the Tcl shell are interpreted when you press Enter. To run a Tcl script in the interactive shell type:

```
source <script name>
```

If a command is not recognized by the shell, it is assumed to be external and executed with the `exec` command.

5.4.1.3 Evaluate as Tcl

Running an executable with the `--tcl_eval` option causes the executable to immediately evaluate the remaining command-line arguments as Tcl commands. This can be useful if you want to run simple Tcl commands from other scripting languages.

For example, the following command runs the Tcl command that prints out the commands available in the project package.

```
quartus_sh --tcl_eval help -pkg project
```

5.4.2 The Quartus Prime Tcl Console Window

To run Tcl commands directly in the Quartus Prime **Tcl Console** window, click **View**. By default, the **Tcl Console** window is docked in the bottom-right corner of the Quartus Prime GUI. All Tcl commands typed in the **Tcl Console** are interpreted by the Quartus Prime Tcl shell.

Note: Some shell commands such as `cd`, `ls`, and others can be run in the Tcl Console window, with the Tcl `exec` command. However, for best results, run shell commands and Quartus Prime executables from a system command prompt outside of the Quartus Prime software GUI.

Tcl messages appear in the **System** tab (**Messages** window). Errors and messages written to `stdout` and `stderr` also are shown in the Quartus Prime **Tcl Console** window.

5.5 End-to-End Design Flows

You can use Tcl scripts to control all aspects of the design flow, including controlling other software, when the other software also includes a scripting interface.

Typically, EDA tools include their own script interpreters that extend core language functionality with tool-specific commands. For example, the Quartus Prime Tcl interpreter supports all core Tcl commands, and adds numerous commands specific to the Quartus Prime software. You can include commands in one Tcl script to run another script, which allows you to combine or chain together scripts to control

different tools. Because scripts for different tools must be executed with different Tcl interpreters, it is difficult to pass information between the scripts unless one script writes information into a file and another script reads it.

Within the Quartus Prime software, you can perform many different operations in a design flow (such as synthesis, fitting, and timing analysis) from a single script, making it easy to maintain global state information and pass data between the operations. However, there are some limitations on the operations you can perform in a single script due to the various packages supported by each executable.

There are no limitations on running flows from any executable. Flows include operations found in

Processing ➤ Start in the Quartus Prime GUI, and are also documented as options for the `execute_flow` Tcl command. If you can make settings in the Quartus Prime software and run a flow to get your desired result, you can make the same settings and run the same flow in a Tcl script.

5.6 Creating Projects and Making Assignments

You can create a script that makes all the assignments for an existing project, and then use the script at any time to restore your project settings to a known state.

Click **Project ➤ Generate Tcl File for Project** to automatically generate a `.tcl` file containing your assignments. You can source this file to recreate your project, and you can add other commands to this file, such as commands for compiling the design. This file is a good starting point to learn about project management and assignment commands.
To commit the assignments you create or modify to the `.qsf` file, you use the `export_assignments` or `project_close` commands. However, when you run the `execute_flow` command, Quartus Prime software automatically commits the assignment changes to the `.qsf` file. To prevent this behavior, specify the `-dont_export_assignments` logic option.

Related Links

- Interactive Shell Mode on page 77
  Running an executable with the `-s` option starts an interactive Tcl shell.
- Constraining Designs on page 9
  Constraints, sometimes known as assignments or logic options, control the way the Quartus Prime software implements a design for an FPGA.
- Quartus Prime Settings File Reference Manual

5.7 Compiling Designs

You can run the Quartus Prime command-line executables from Tcl scripts. Use the included `flow` package to run various Quartus Prime compilation flows, or run each executable directly.

5.7.1 The flow Package

The `flow` package includes two commands for running Quartus Prime command-line executables, either individually or together in standard compilation sequence.

- The `execute_module` command allows you to run an individual Quartus Prime command-line executable.
- The `execute_flow` command allows you to run some or all of the executables in commonly-used combinations.

Use the `flow` package instead of system calls to run Quartus Prime executables from scripts or from the Quartus Prime Tcl Console.

5.7.2 Compile All Revisions

You can use a simple Tcl script to compile all revisions in your project. Save the following script in a file called `compile_revisions.tcl` and type the following to run it:

```
quartus_sh -t compile_revisions.tcl <project name>
```

Compile All Revisions

```
load_package flow
project_open [lindex $quartus(args) 0]
set original_revision [get_current_revision]
foreach revision [get_project_revisions] {
    set_current_revision $revision
    execute flow -compile
}
set_current_revision $original_revision
project_close
```

5.8 Reporting

You can extract information from the Compilation Report to evaluate results. The Quartus Prime Tcl API provides easy access to report data so you do not have to write scripts to parse the text report files.

If you know the exact report cell or cells you want to access, use the `get_report_panel_data` command and specify the row and column names (or x and y coordinates) and the name of the appropriate report panel. You can often search for data in a report panel. To do this, use a loop that reads the report one row at a time with the `get_report_panel_row` command.

Column headings in report panels are in row 0. If you use a loop that reads the report one row at a time, start with row 1 to skip column headings. The `get_number_of_rows` command returns the number of rows in the report panel, including the column heading row. Since the number of rows includes the column heading row, continue your loop if the loop index is less than the number of rows.

Report panels are hierarchically arranged and each level of hierarchy is denoted by the string "||" in the panel name. For example, the name of the Fitter Settings report panel is `Fitter||Fitter Settings` because it is in the `Fitter` folder. Panels at the highest hierarchy level do not use the "||" string. For example, the Flow Settings report panel is named `Flow Settings`.

The following Tcl code prints a list of all report panel names in your project. You can run this code with any executable that includes support for the report package.

Print All Report Panel Names

```
load_package report
project_open myproject
load_report
set panel_names [get_report_panel_names]
foreach panel_name $panel_names {
post_message "$panel_name"
}
```

5.8.1 Viewing Report Data in Excel

The Microsoft Excel software can be useful in viewing and manipulating timing analysis results. You can create a Comma Separated Value (`.csv`) file from any Quartus Prime report to open with Excel. The following Tcl code shows a simple way to create a `.csv` file with data from the Fitter panel in a report. You could modify the script to use command-line arguments to pass in the name of the project, report panel, and output file to use. You can run this script example with any executable that supports the report package.

Create .csv Files from Reports

```
load_package report
project_open my-project
load_report
# This is the name of the report panel to save as a CSV file
set panel_name "Fitter||Fitter Settings"
set csv_file "output.csv"
set fh [open $csv_file w]
set num_rows [get_number_of_rows -name $panel_name]
# Go through all the rows in the report file, including the
# row with headings, and write out the comma-separated data
for { set i 0 } { $i < $num_rows } { incr i } {
    set row_data [get_report_panel_row -name $panel_name \
        -row $i]
    puts $fh [join $row_data ","]
}
close $fh
unload_report
```

5.9 Timing Analysis

The Quartus Prime TimeQuest Timing Analyzer includes support for industry-standard SDC commands in the `sdc` package.

The Quartus Prime software includes comprehensive Tcl APIs and SDC extensions for the TimeQuest Timing Analyzer in the `sta`, and `sdc_ext` packages. The Quartus Prime software also includes a `tdc` package that obtains information from the TimeQuest Timing Analyzer.

Related Links

Quartus Prime Pro Edition Settings File Reference Manual
For information about all settings and constraints in the Quartus Prime software.

5.10 Automating Script Execution

You can configure scripts to run automatically at various points during compilation. Use this capability to automatically run scripts that perform custom reporting, make specific assignments, and perform many other tasks.

The following three global assignments control when a script is run automatically:

- PRE_FLOW_SCRIPT_FILE —before a flow starts
- POST_MODULE_SCRIPT_FILE —after a module finishes
- POST_FLOW_SCRIPT_FILE —after a flow finishes

A module is another term for a Quartus Prime executable that performs one step in a flow. For example, two modules are Analysis and Synthesis (`quartus_syn`), and timing analysis (`quartus_sta`).

A flow is a series of modules that the Quartus Prime software runs with predefined options. For example, compiling a design is a flow that typically consists of the following steps (performed by the indicated module):

1. Analysis and Synthesis (`quartus_syn`)
2. Fitter (`quartus_fit`)
3. Assembler (`quartus_asm`)
4. Timing Analyzer (`quartus_sta`)

Other flows are described in the help for the `execute_flow` Tcl command. In addition, many commands in the **Processing** menu of the Quartus Prime GUI correspond to this design flow.

To make an assignment automatically run a script, add an assignment with the following form to the `.qsf` for your project:

```
set_global_assignment -name <assignment name> <executable>:<script name>
```

The Quartus Prime software runs the scripts.

```
<executable> -t <script name> <flow or module name> <project name> <revision name>
```

The first argument passed in the `argv` variable (or `quartus(args)` variable) is the name of the flow or module being executed, depending on the assignment you use. The second argument is the name of the project and the third argument is the name of the revision.

When you use the POST_MODULE_SCRIPT_FILE assignment, the specified script is automatically run after every executable in a flow. You can use a string comparison with the module name (the first argument passed in to the script) to isolate script processing to certain modules.

5.10.1 Execution Example

To illustrate how automatic script execution works in a complete flow, assume you have a project called **top** with a current revision called **rev_1**, and you have the following assignments in the `.qsf` for your project.

```
set_global_assignment -name PRE_FLOW_SCRIPT_FILE quartus_sh:first.tcl
set_global_assignment -name POST_MODULE_SCRIPT_FILE quartus_sh:next.tcl
set_global_assignment -name POST_FLOW_SCRIPT_FILE quartus_sh:last.tcl
```

When you compile your project, the PRE_FLOW_SCRIPT_FILE assignment causes the following command to be run before compilation begins:

```
quartus_sh -t first.tcl compile top rev_1
```

Next, the Quartus Prime software starts compilation with analysis and synthesis, performed by the `quartus_syn` executable. After the Analysis and Synthesis finishes, the POST_MODULE_SCRIPT_FILE assignment causes the following command to run:

```
quartus_sh -t next.tcl quartus_syn top rev_1
```

Then, the Quartus Prime software continues compilation with the Fitter, performed by the `quartus_fit` executable. After the Fitter finishes, the POST_MODULE_SCRIPT_FILE assignment runs the following command:

```
quartus_sh -t next.tcl quartus_fit top rev_1
```

Corresponding commands are run after the other stages of the compilation. When the compilation is over, the POST_FLOW_SCRIPT_FILE assignment runs the following command:

```
quartus_sh -t last.tcl compile top rev_1
```

5.10.2 Controlling Processing

The POST_MODULE_SCRIPT_FILE assignment causes a script to run after every module. Because the same script is run after every module, you might have to include some conditional statements that restrict processing in your script to certain modules.

For example, if you want a script to run only after timing analysis, use a conditional test like the following example. It checks the flow or module name passed as the first argument to the script and executes code when the module is `quartus_sta`.

Restrict Processing to a Single Module

```
set module [lindex $quartus(args) 0]
if [string match "quartus_sta" $module] {
    # Include commands here that are run
    # after timing analysis
    # Use the post-message command to display
    # messages
    post_message "Running after timing analysis"
}
```

5.10.3 Displaying Messages

Because of the way the Quartus Prime software runs the scripts automatically, you must use the `post_message` command to display messages, instead of the `puts` command. This requirement applies only to scripts that are run by the three assignments listed in "Automating Script Execution".

Related Links

- The post_message Command on page 85
  To print messages that are formatted like Quartus Prime software messages, use the `post_message` command.
- Automating Script Execution on page 81
  You can configure scripts to run automatically at various points during compilation.

5.11 Other Scripting Features

The Quartus Prime Tcl API includes other general-purpose commands and features described in this section.

5.11.1 Natural Bus Naming

The Quartus Prime software supports natural bus naming. Natural bus naming allows to use square brackets to specify bus indexes in HDL without including escape characters to prevent Tcl from interpreting the square brackets as containing commands. For example, one signal in a bus named `address` can be identified as `address[0]` instead of `address\[0\]`. You can take advantage of natural bus naming when making assignments.

```
set_location_assignment -to address[10] Pin_M20
```

The Quartus Prime software defaults to natural bus naming. You can turn off natural bus naming with the `disable_natural_bus_naming` command. For more information about natural bus naming, type the following at a Quartus Prime Tcl prompt:

```
enable_natural_bus_naming -h
```

5.11.2 Short Option Names

You can use short versions of command options, as long as they are unambiguous. For example, the `project_open` command supports two options: `-current_revision` and `-revision`.

You can use any of the following abbreviations of the `-revision` option:

- `-r`
- `-re`
- `-rev`
- `-revi`
- `-revis`
- `-revisio`

You can use an option as short as `-r` because in the case of the `project_open` command no other option starts with the letter r. However, the `report_timing` command includes the options `-recovery` and `-removal`. You cannot use `-r` or `-re` to shorten either of those options, because the abbreviation would not be unique to only one option.

5.11.3 Collection Commands

Some Quartus Prime Tcl functions return very large sets of data that would be inefficient as Tcl lists. These data structures are referred to as collections. The Quartus Prime Tcl API uses a collection ID to access the collection.

There are two Quartus Prime Tcl commands for working with collections, `foreach_in_collection` and `get_collection_size`. Use the `set` command to assign a collection ID to a variable.

5.11.3.1 The foreach_in_collection Command

The `foreach_in_collection` command is similar to the `foreach` Tcl command. Use it to iterate through all elements in a collection. The following example prints all instance assignments in an open project.

foreach_in_collection Example

```
set all_instance_assignments [get_all_instance_assignments -name *]
foreach_in_collection asgn $all_instance_assignments {
    # Information about each assignment is
    # returned in a list. For information
    # about the list elements, refer to Help
    # for the get-all-instance-assignments command.
    set to [lindex $asgn 2]
    set name [lindex $asgn 3]
    set value [lindex $asgn 4]
    puts "Assignment to $to: $name = $value"
}
```

Related Links

foreach_in_collection (::quartus::misc)
In Quartus Prime Help

5.11.3.2 The get_collection_size Command

Use the `get_collection_size` command to get the number of elements in a collection. The following example prints the number of global assignments in an open project.

get_collection_size Example

```
set all_global_assignments [get_all_global_assignments -name *]
set num_global_assignments [get_collection_size $all_global_assignments]
puts "There are $num_global_assignments global assignments in your project"
```

5.11.4 The post_message Command

To print messages that are formatted like Quartus Prime software messages, use the `post_message` command. Messages printed by the `post_message` command appear in the **System** tab of the **Messages** window in the Quartus Prime GUI, and are written to standard at when scripts are run. Arguments for the `post_message` command include an optional message type and a required message string.

The message type can be one of the following:

- `info` (default)
- `extra_info`
- `warning`
- `critical_warning`
- `error`

If you do not specify a type, Quartus Prime software defaults to `info`.

With the Quartus Prime software in Windows, you can color code messages displayed at the system command prompt with the `post_message` command. Add the following line to your `quartus2.ini` file:

```
DISPLAY_COMMAND_LINE_MESSAGES_IN_COLOR = on
```

The following example shows how to use the `post_message` command.

```
post_message -type warning "Design has gated clocks"
```

5.11.5 Accessing Command-Line Arguments

The global variable `quartus(args)` is a list of the arguments typed on the command-line following the name of the Tcl script.

Example 4. Simple Command-Line Argument Access

The following Tcl example prints all the arguments in the `quartus(args)` variable:

```
set i 0
foreach arg $quartus(args) {
    puts "The value at index $i is $arg"
    incr i
}
```

Example 5. Passing Command-Line Arguments to Scripts

If you copy the script in the previous example to a file named `print_args.tcl`, it displays the following output when you type the following at a command prompt.

```
quartus_sh -t print_args.tcl my_project 100MHz
The value at index 0 is my_project
The value at index 1 is 100MHz
```

5.11.5.1 The cmdline Package

You can use the `cmdline` package included with the Quartus Prime software for more robust and self-documenting command-line argument passing. The `cmdline` package supports command-line arguments with the form `-<option><value>`.

cmdline Package

```
package require cmdline
variable ::argv0 $::quartus(args)
set options {
   { "project.arg" "" "Project name" }
   { "frequency.arg" "" "Frequency" }
}
set usage "You need to specify options and values"
array set optshash [::cmdline::getoptions ::argv $options $usage]
puts "The project name is $optshash(project)"
puts "The frequency is $optshash(frequency)"
```

If you save those commands in a Tcl script called `print_cmd_args.tcl` you see the following output when you type the following command at a command prompt.

Passing Command-Line Arguments for Scripts

```
quartus_sh -t print_cmd_args.tcl -project my_project -frequency 100MHz
The project name is my_project
The frequency is 100MHz
```

Virtually all Quartus Prime Tcl scripts must open a project. You can open a project, and you can optionally specify a revision name with code like the following example. The example checks whether the specified project exists. If it does, the example opens the current revision, or the revision you specify.

Full-Featured Method to Open Projects

```
package require cmdline
variable ::argv0 $::quartus(args)
set options { \
{ "project.arg" "" "Project Name" } \
{ "revision.arg" "" "Revision Name" } \
}
array set optshash [::cmdline::getoptions ::argv0 $options]
# Ensure the project exists before trying to open it
if {[project_exists $optshash(project)]} {
    if {[string equal "" $optshash(revision)]} {
        # There is no revision name specified, so default
        # to the current revision
        project_open $optshash(project) -current_revision
    } else {
        # There is a revision name specified, so open the
```

```
        # project with that revision
        project_open $optshash(project) -revision \
            $optshash(revision)
    }
} else {
    puts "Project $optshash(project) does not exist"
    exit 1
}
# The rest of your script goes here
```

If you do not require this flexibility or error checking, you can use just the `project_open` command.

Simple Method to Open Projects

```
set proj_name [lindex $argv 0]
project_open $proj_name
```

5.11.6 The quartus() Array

The global `quartus()` Tcl array includes other information about your project and the current Quartus Prime executable that might be useful to your scripts. The scripts in the preceding examples parsed command line arguments found in `quartus(args)`. For information on the other elements of the `quartus()` array, type the following command at a Tcl prompt:

```
help -quartus
```

5.12 The Quartus Prime Tcl Shell in Interactive Mode Example

This section presents how to make project assignments and then compile the finite impulse response (FIR) filter tutorial project with the `quartus_sh` interactive shell.

This example assumes you already have the `fir_filter` tutorial design files in a project directory.

1. To run the interactive Tcl shell, type the following at the system command prompt:

   ```
   quartus_sh -s
   ```

2. Create a new project called `fir_filter`, with a revision called `filtref` by typing:

   ```
   project_new -revision filtref fir_filter
   ```

   Note: If the project file and project name are the same, the Quartus Prime software gives the revision the same name as the project.

   Because the revision named `filtref` matches the top-level file, all design files are automatically picked up from the hierarchy tree.

3. Set a global assignment for the device:

   ```
   set_global_assignment -name family <device family name>
   ```

To learn more about assignment names that you can use with the `-name` option, refer to Quartus Prime Help.

Note: For assignment values that contain spaces, enclose the value in quotation marks.

4. To compile a design, use the `::quartus::flow` package, which properly exports the new project assignments and compiles the design with the proper sequence of the command-line executables. First, load the package:

```
load_package flow
```

It returns:

```
1.1
```

5. To perform a full compilation of the FIR filter design, use the `execute_flow` command with the `-compile` option:

```
execute_flow -compile
```

This command compiles the FIR filter tutorial project, exporting the project assignments and running `quartus_syn`, `quartus_fit`, `quartus_asm`, and `quartus_sta`. This sequence of events is the same as selecting **Processing ➤ Start Compilation** in the Quartus Prime GUI.

6. When you are finished with a project, close it with the `project_close` command.
7. To exit the interactive Tcl shell, type `exit` at a Tcl prompt.

Related Links

set_global_assignment (::quartus::project)
In Quartus Prime Help

5.13 The tclsh Shell

On the UNIX and Linux operating systems, the tclsh shell included with the Quartus Prime software is initialized with a minimal PATH environment variable. As a result, system commands might not be available within the tclsh shell because certain directories are not in the PATH environment variable.

To include other directories in the path searched by the tclsh shell, set the `QUARTUS_INIT_PATH` environment variable before running the tclsh shell. Directories in the `QUARTUS_INIT_PATH` environment variable are searched by the tclsh shell when you execute a system command.

5.14 Tcl Scripting Basics

The core Tcl commands support variables, control structures, and procedures. Additionally, there are commands for accessing the file system and network sockets, and running other programs. You can create platform-independent graphical interfaces with the Tk widget set.

Tcl commands are executed immediately as they are typed in an interactive Tcl shell. You can also create scripts (including the examples in this chapter) in files and run them with the Quartus Prime executables or with the tclsh shell.

5.14.1 Hello World Example

The following shows the basic "Hello world" example in Tcl:

```
puts "Hello world"
```

Use double quotation marks to group the words `hello` and `world` as one argument. Double quotation marks allow substitutions to occur in the group. Substitutions can be simple variable substitutions, or the result of running a nested command. Use curly braces {} for grouping when you want to prevent substitutions.

5.14.2 Variables

Assign a value to a variable with the `set` command. You do not have to declare a variable before using it. Tcl variable names are case-sensitive.

```
set a 1
```

To access the contents of a variable, use a dollar sign ("$") before the variable name. The following example prints `"Hello world"` in a different way.

```
set a Hello
set b world
puts "$a $b"
```

5.14.3 Substitutions

Tcl performs three types of substitution:

- Variable value substitution
- Nested command substitution
- Backslash substitution

5.14.3.1 Variable Value Substitution

Variable value substitution, refers to accessing the value stored in a variable with a dollar sign ("$") before the variable name.

5.14.3.2 Nested Command Substitution

Nested command substitution refers to how the Tcl interpreter evaluates Tcl code in square brackets. The Tcl interpreter evaluates nested commands, starting with the innermost nested command, and commands nested at the same level from left to right. Each nested command result is substituted in the outer command.

```
set a [string length foo]
```

5.14.3.3 Backslash Substitution

Backslash substitution allows you to quote reserved characters in Tcl, such as dollar signs ("$") and braces ("[ ]"). You can also specify other special ASCII characters like tabs and new lines with backslash substitutions. The backslash character is the Tcl line continuation character, used when a Tcl command wraps to more than one line.

```
set this_is_a_long_variable_name [string length "Hello \
    world."]
```

5.14.4 Arithmetic

Use the `expr` command to perform arithmetic calculations. Use curly braces ("{ }") to group the arguments of this command for greater efficiency and numeric precision.

```
set a 5
set b [expr { $a + sqrt(2) }]
```

Tcl also supports boolean operators such as && (AND), || (OR), ! (NOT), and comparison operators such as < (less than), > (greater than), and == (equal to).

5.14.5 Lists

A Tcl list is a series of values. Supported list operations include creating lists, appending lists, extracting list elements, computing the length of a list, sorting a list, and more.

```
set a { 1 2 3 }
```

You can use the `lindex` command to extract information at a specific index in a list. Indexes are zero-based. You can use the index `end` to specify the last element in the list, or the index `end-`<*n*> to count from the end of the list. For example, to print the second element (at index 1) in the list stored in a use the following code.

```
puts [lindex $a 1]
```

The `llength` command returns the length of a list.

```
puts [llength $a]
```

The `lappend` command appends elements to a list. If a list does not already exist, the list you specify is created. The list variable name is not specified with a dollar sign ("$").

```
lappend a 4 5 6
```

5.14.6 Arrays

Arrays are similar to lists except that they use a string-based index. Tcl arrays are implemented as hash tables. You can create arrays by setting each element individually or with the `array set` command.

To set an element with an index of `Mon` to a value of `Monday` in an array called `days`, use the following command:

```
set days(Mon) Monday
```

The `array set` command requires a list of index/value pairs. This example sets the array called days:

```
array set days { Sun Sunday Mon Monday Tue Tuesday \
    Wed Wednesday Thu Thursday Fri Friday Sat Saturday }
```

```
set day_abbreviation Mon
puts $days($day_abbreviation)
```

Use the `array names` command to get a list of all the indexes in a particular array. The index values are not returned in any specified order. The following example is one way to iterate over all the values in an array.

```
foreach day [array names days] {
    puts "The abbreviation $day corresponds to the day \
name $days($day)"
}
```

Arrays are a very flexible way of storing information in a Tcl script and are a good way to build complex data structures.

5.14.7 Control Structures

Tcl supports common control structures, including if-then-else conditions and `for`, `foreach`, and `while` loops. The position of the curly braces as shown in the following examples ensures the control structure commands are executed efficiently and correctly. The following example prints whether the value of variable a positive, negative, or zero.

If-Then-Else Structure

```
if { $a > 0 } {
    puts "The value is positive"
} elseif { $a < 0 } {
    puts "The value is negative"
} else {
    puts "The value is zero"
}
```

The following example uses a `for` loop to print each element in a list.

For Loop

```
set a { 1 2 3 }
for { set i 0 } { $i < [llength $a] } { incr i } {
    puts "The list element at index $i is [lindex $a $i]"
}
```

The following example uses a `foreach` loop to print each element in a list.

foreach Loop

```
set a { 1 2 3 }
foreach element $a {
    puts "The list element is $element"
}
```

The following example uses a `while` loop to print each element in a list.

while Loop

```
set a { 1 2 3 }
set i 0
while { $i < [llength $a] } {
    puts "The list element at index $i is [lindex $a $i]"
    incr i
}
```

You do not have to use the `expr` command in boolean expressions in control structure commands because they invoke the `expr` command automatically.

5.14.8 Procedures

Use the `proc` command to define a Tcl procedure (known as a subroutine or function in other scripting and programming languages). The scope of variables in a procedure is local to the procedure. If the procedure returns a value, use the `return` command to return the value from the procedure. The following example defines a procedure that multiplies two numbers and returns the result.

Simple Procedure

```
proc multiply { x y } {
    set product [expr { $x * $y }]
    return $product
}
```

The following example shows how to use the `multiply` procedure in your code. You must define a procedure before your script calls it.

Using a Procedure

```
proc multiply { x y } {
    set product [expr { $x * $y }]
    return $product
}
```

```
set a 1
set b 2
puts [multiply $a $b]
```

Define procedures near the beginning of a script. If you want to access global variables in a procedure, use the `global` command in each procedure that uses a global variable.

Accessing Global Variables

```
proc print_global_list_element { i } {
    global my_data
    puts "The list element at index $i is [lindex $my_data $i]"
}
set my_data { 1 2 3}
print_global_list_element 0
```

5.14.9 File I/O

Tcl includes commands to read from and write to files. You must open a file before you can read from or write to it, and close it when the read and write operations are done.

To open a file, use the `open` command; to close a file, use the `close` command. When you open a file, specify its name and the mode in which to open it. If you do not specify a mode, Tcl defaults to read mode. To write to a file, specify `w` for write mode.

Open a File for Writing

```
set output [open myfile.txt w]
```

Tcl supports other modes, including appending to existing files and reading from and writing to the same file.

The `open` command returns a file handle to use for read or write access. You can use the `puts` command to write to a file by specifying a file handle.

Write to a File

```
set output [open myfile.txt w]
puts $output "This text is written to the file."
close $output
```

You can read a file one line at a time with the `gets` command. The following example uses the `gets` command to read each line of the file and then prints it out with its line number.

Read from a File

```
set input [open myfile.txt]
set line_num 1
while { [gets $input line] >= 0 } {
    # Process the line of text here
    puts "$line_num: $line"
    incr line_num
}
close $input
```

5.14.10 Syntax and Comments

Arguments to Tcl commands are separated by white space, and Tcl commands are terminated by a newline character or a semicolon. You must use backslashes when a Tcl command extends more than one line.

Tcl uses the hash or pound character (#) to begin comments. The # character must begin a comment. If you prefer to include comments on the same line as a command, be sure to terminate the command with a semicolon before the # character. The following example is a valid line of code that includes a `set` command and a comment.

```
set a 1;# Initializes a
```

Without the semicolon, it would be an invalid command because the `set` command would not terminate until the new line after the comment.

The Tcl interpreter counts curly braces inside comments, which can lead to errors that are difficult to track down. The following example causes an error because of unbalanced curly braces.

```
# if { $x > 0 } {
if { $y > 0 } {
    # code here
}
```

5.14.11 External References

For more information about Tcl, refer to the following sources:

- Brent B. Welch and Ken Jones, and Jeffery Hobbs, *Practical Programming in Tcl and Tk* (Upper Saddle River: Prentice Hall, 2003)
- John Ousterhout and Ken Jones, *Tcl and the Tk Toolkit* (Boston: Addison-Wesley Professional, 2009)
- Mark Harrison and Michael McLennan, *Effective Tcl/Tk Programming: Writing Better Programs in Tcl and Tk* (Boston: Addison-Wesley Professional, 1997)

Related Links

- Quartus Prime Tcl Examples
  For Quartus Prime Tcl example scripts
- tcl.activestate.com
  Tcl Developer Xchange

5.15 Document Revision History

Table 31. Document Revision History

Date	Version	Changes
2016.10.31	16.1.0	• Implemented Intel rebranding.
2015.11.02	15.1.0	• Changed instances of *Quartus II* to *Quartus Prime*. • Updated the list of Tcl packages in the *Quartus Prime Tcl Packages* section. • Updated the *Quartus Prime Tcl API Help* section: — Updated the Tcl Help Output
June 2014	14.0.0	Updated the format.
June 2012	12.0.0	• Removed survey link.
November 2011	11.0.1	• Template update • Updated supported version of Tcl in the section "Tool Command Language." • minor editoral changes
May 2011	11.0.0	Minor updates throughout document.
December 2010	10.1.0	Template update Updated to remove tcl packages used by the Classic Timing Analyzer
July 2010	10.0.0	Minor updates throughout document.
November 2009	9.1.0	• Removed LogicLock example. • Added the incremental_compilation, insystem_source_probe, and rtl packages to Table 3-1 and Table 3-2. • Added quartus_map to table 3-2.
March 2009	9.0.0	• Removed the "EDA Tool Assignments" section • Added the section "Compile All Revisions" on page 3–9 • Added the section "Using the tclsh Shell" on page 3–20
November 2008	8.1.0	Changed to 8½" × 11" page size. No change to content.
May 2008	8.0.0	Updated references.

Related Links

Altera Documentation Archive

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

6 Reviewing Printed Circuit Board Schematics with the Quartus Prime Software

Intel FPGAs and CPLDs offer a multitude of configurable options to allow you to implement a custom application-specific circuit on your PCB.

Your Quartus Prime project provides important information specific to your programmable logic design, which you can use in conjunction with the device literature available on Altera's website to ensure that you implement the correct board-level connections in your schematic.

Refer to the **Settings** dialog box options, the Fitter report, and **Messages** window when creating and reviewing your PCB schematic. The Quartus Prime software also provides the Pin Planner to assist you during your PCB schematic review process.

Related Links

- Schematic Review Worksheets
- Pin Connection Guidelines

6.1 Reviewing Quartus Prime Software Settings

Review these settings in the Quartus Prime software to help you review your PCB schematic.

The **Device** dialog box in the Quartus Prime software allows you to specify device-specific assignments and settings. You can use the **Device** dialog box to specify general project-wide options, including specific device and pin options, which help you to implement correct board-level connections in your PCB schematic.

The **Device** dialog box provides project-specific device information, including the target device and any migration devices you specify. Using migration devices can impact the number of available user I/O pins and internal resources, as well as require connection of some user I/O pins to power/ground pins to support migration.

If you want to use vertical migration, which allows you to use different devices with the same package, you can specify your list of migration devices in the **Migration Devices** dialog box. The Fitter places the pins in your design based on your targeted migration devices, and allows you to use only I/O pins that are common to all of the migration devices.

If a migration device has pins that are power or ground, but the pins are also user I/O pins on a different device in the migration path, the Fitter ensures that these pins are not used as user I/O pins. You must ensure that these pins are connected to the appropriate plane on the PCB.

If you are migrating from a smaller device with NC (no-connect) pins to a larger device with power or ground pins in the same package, you can safely connect the NC pins to power or ground pins to facilitate successful migration.

Related Links

Migration Devices Dialog Box
In Quartus Prime Help

6.1.1 Device and Pins Options Dialog Box Settings

You can set device and pin options and verify important design-specific data in the **Device and Pin Options** dialog box, including options found on the **General**, **Configuration**, **Unused Pin**, **Dual-Purpose Pins**, and **Voltage** pages.

6.1.1.1 Configuration Settings

The **Configuration** page of the **Device and Pin Options** dialog box specifies the configuration scheme and configuration device for the target device. Use the **Configuration** page settings to verify the configuration scheme with the MSEL pin settings used on your PCB schematic and the I/O voltage of the configuration scheme.

Your specific configuration settings may impact the availability of some dual-purpose I/O pins in user mode.

Related Links

Dual-Purpose Pins Settings on page 97

6.1.1.2 Unused Pin Settings

The **Unused Pin** page specifies the behavior of all unused pins in your design. Use the **Unused Pin** page to ensure that unused pin settings are compatible with your PCB.

For example, if you reserve all unused pins as outputs driving ground, you must ensure that you do not connect unused I/O pins to VCC pins on your PCB. Connecting unused I/O pins to VCC pins may result in contention that could lead to higher than expected current draw and possible device overstress.

The **Reserve all unused pins** list shows available unused pin state options for the target device. The default state for each pin is the recommended setting for each device family.

When you reserve a pin as output driving ground, the Fitter connects a ground signal to the output pin internally. You should connect the output pin to the ground plane on your PCB, although you are not required to do so. Connecting the output driving ground to the ground plane is known as creating a virtual ground pin, which helps to minimize simultaneous switching noise (SSN) and ground bounce effects.

6.1.1.3 Dual-Purpose Pins Settings

The **Dual-Purpose Pins** page specifies how configuration pins should be used after device configuration completes. You can set the function of the dual-purpose pins by selecting a value for a specific pin in the **Dual-purpose pins** list. Pin functions should match your PCB schematic. The available options on the **Dual-Purpose Pins** page may differ depending on the selected configuration mode.

6.1.1.4 Voltage Settings

The **Voltage** page specifies the default VCCIO I/O bank voltage and the default I/O bank voltage for the pins on the target device. VCCIO I/O bank voltage settings made in the **Voltage** page are overridden by I/O standard assignments made on I/O pins in their respective banks.

Related Links

Reviewing Device Pin-Out Information in the Fitter Report on page 98
After you compile your design, you can use the reports in the Resource section of the Fitter report to check your device pin-out in detail.

6.1.1.5 Error Detection CRC Settings

The **Error Detection CRC** page specifies error detection cyclic redundancy check (CRC) use for the target device. When **Enable error detection CRC** is turned on, the device checks the validity of the programming data in the devices. Any changes made in the data while the device is in operation generates an error.

Turning on the **Enable open drain on CRC error pin** option allows the CRC ERROR pin to be set as an open-drain pin in some devices, which decouples the voltage level of the CRC ERROR pin from VCCIO voltage. You must connect a pull-up resistor to the CRC ERROR pin on your PCB if you turn on this option.

In addition to settings in the **Device** dialog box, you should verify settings in the **Voltage** page of the **Settings** dialog box.

Related Links

Device and Pin Options Dialog Box
In Quartus Prime Help

6.1.2 Voltage Settings

The **Voltage** page, under **Operating Settings and Conditions** in the **Settings** dialog box, allows you to specify voltage operating conditions for timing and power analyses.

Ensure that the settings in the **Voltage** page match the settings in your PCB schematic, especially if the target device includes transceivers.

The **Voltage** page settings requirements differ depending on the settings of the transceiver instances in the design. Refer to the Fitter report for the required settings, and verify that the voltage settings are correctly set up for your PCB schematic.

After verifying your settings in the **Device** and **Settings** dialog boxes, you can verify your device pin-out with the Fitter report.

Related Links

Pin Connection Guidelines

6.2 Reviewing Device Pin-Out Information in the Fitter Report

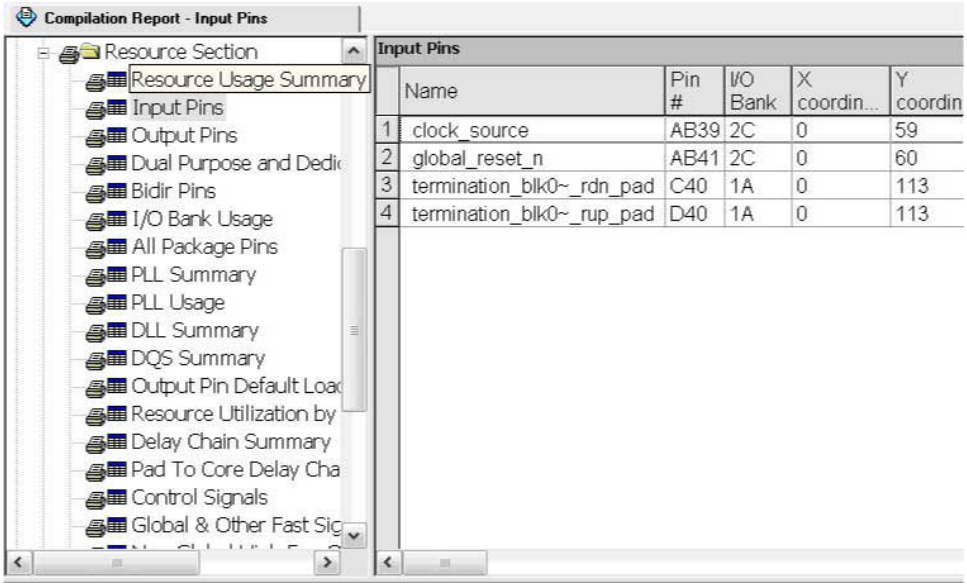
After you compile your design, you can use the reports in the Resource section of the Fitter report to check your device pin-out in detail.

The Input Pins, Output Pins, and Bidirectional Pins reports identify all the user I/O pins in your design and the features enabled for each I/O pin. For example, you can find use of weak internal pull-ups, PCI clamp diodes, and on-chip termination (OCT) pin assignments in these sections of the Fitter report. You can check the pin assignments reported in the Input Pins, Output Pins, and Bidirectional Pins reports against your PCB schematic to determine whether your PCB requires external components.

These reports also identify whether you made pin assignments or if the Fitter automatically placed the pins. If the Fitter changed your pin assignments, you should make these changes user assignments because the location of pin assignments made by the Fitter may change with subsequent compilations.

Figure 34. Resource Section Report

This figure shows the pins the Fitter chose for the OCT external calibration resistor connections (RUP/RDN) and the name of the associated termination block in the Input Pins report. You should make these types of assignments user assignments.



The I/O Bank Usage report provides a high-level overview of the VCCIO and VREF requirements for your design, based on your I/O assignments. Verify that the requirements in this report match the settings in your PCB schematic. All unused I/O banks, and all banks with I/O pins with undefined I/O standards, default the VCCIO voltage to the voltage defined in the **Voltage** page of the **Device and Pin Options** dialog box.

The All Package Pins report lists all the pins on your device, including unused pins, dedicated pins and power/ground pins. You can use this report to verify pin characteristics, such as the location, name, usage, direction, I/O standard and voltage for each pin with the pin information in your PCB schematic. In particular, you should verify the recommended voltage levels at which you connect unused dedicated inputs and I/O and power pins, especially if you selected a migration device. Use the All Package Pins report to verify that you connected all the device voltage rails to the voltages reported.

Errors commonly reported include connecting the incorrect voltage to the predriver supply (VCCPD) pin in a specific bank, or leaving dedicated clock input pins floating. Unused input pins that should be connected to ground are designated as **GND+** in the **Pin Name/Usage** column in the All Package Pins report.

You can also use the All Package Pins report to check transceiver-specific pin connections and verify that they match the PCB schematic. Unused transceiver pins have the following requirements, based on the pin designation in the Fitter report:

- GXB_GND—Unused GXB receiver or dedicated reference clock pin. This pin must be connected to GXB_GND through a 10k Ohm resistor.
- GXB_NC—Unused GXB transmitter or dedicated clock output pin. This pin must be disconnected.

Some transceiver power supply rails have dual voltage capabilities, such as VCCA_L/R and VCCH_L/R, that depend on the settings you created for the ALTGX parameter editor. Because these user-defined settings overwrite the default settings, you should use the All Package Pins report to verify that these power pins on the device symbol in the PCB schematics are connected to the voltage required by the transceiver. An incorrect connection may cause the transceiver to function not as expected.

If your design includes a memory interface, the DQS Summary report provides an overview of each DQ pin group. You can use this report to quickly confirm that the correct DQ/DQS pins are grouped together.

Finally, the Fitter Device Options report summarizes some of the settings made in the **Device and Pin Options** dialog box. Verify that these settings match your PCB schematics.

6.3 Reviewing Compilation Error and Warning Messages

If your project does not compile without error or warning messages, you should resolve the issues identified by the Compiler before signing off on your pin-out or PCB schematic. Error messages often indicate illegal or unsupported use of the device resources and IP.

Additionally, you should cross-reference fitting and timing analysis warnings with the design implementation. Timing may be constrained due to nonideal pin placement. You should investigate if you can reassign pins to different locations to prevent fitting and timing analysis warnings. Ensure that you review each warning and consider its potential impact on the design.

6.4 Using Additional Quartus Prime Software Features

You can generate IBIS files, which contain models specific to your design and selected I/O standards and options, with the Quartus Prime software.

Because board-level simulation is important to verify, you should check for potential signal integrity issues. You can turn on the **Board-Level Signal Integrity** feature in the **EDA Tool Settings** page of the **Settings** dialog box.

Additionally, using advanced I/O timing allows you to enter physical PCB information to accurately model the load seen by an output pin. This feature facilitates accurate I/O timing analysis.

Related Links

- Signal Integrity Analysis with Third-Party Tools on page 221
- Managing Device I/O Pins on page 18

6.5 Using Additional Quartus Prime Software Tools

Use the Pin Planner to assist you with reviewing your PCB schematics.

6.5.1 Pin Planner

The Quartus Prime Pin Planner helps you visualize, plan, and assign device I/O pins in a graphical view of the target device package. You can quickly locate various I/O pins and assign them design elements or other properties to ensure compatibility with your PCB layout.

You can use the Pin Planner to verify the location of clock inputs, and whether they have been placed on dedicated clock input pins, which is recommended when your design uses PLLs.

You can also use the Pin Planner to verify the placement of dedicated SERDES pins. SERDES receiver inputs can be placed only on DIFFIO_RX pins, while SERDES transmitter outputs can be placed only on DIFFIO_TX pins.

The Pin Planner gives a visual indication of signal-to-signal proximity in the **Pad View** window, and also provides information about differential pin pair placement, such as the placement of pseudo-differential signals.

Related Links

Managing Device I/O Pins on page 18

6.6 Document Revision History

Table 32. Document Revision History

Date	Version	Changes
2016.10.31	16.1.0	• Implemented Intel rebranding.
2015.11.02	15.1.0	Changed instances of *Quartus II* to *Quartus Prime*.
June 2014	14.0.0	Template update.
November 2012	12.1.0	Minor update of Pin Planner description for task and report windows.
June 2012	12.0.0	Removed survey link.
November 2011	10.0.2	Template update.
December 2010	10.0.1	Changed to new document template. No change to content.
July 2010	10.0.0	Initial release.

Related Links

Altera Documentation Archive

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

7 Design Optimization Overview

7.1 Design Optimization Overview

This chapter introduces features in the Intel Quartus Prime software that you can use to achieve the highest design performance when you design for programmable logic devices (PLDs), especially high density FPGAs.

Physical implementation can be an intimidating and challenging phase of the design process. The Quartus Prime software provides a comprehensive environment for FPGA designs, delivering unmatched performance, efficiency, and ease-of-use.

In a typical design flow, you must synthesize your design with Quartus Prime Pro Edition synthesis or a third-party tool, place and route your design with the Fitter, and use the TimeQuest timing analyzer to ensure your design meets the timing requirements. With the Power Analyzer, you ensure the design's power consumption is within limits.

7.2 Initial Compilation: Required Settings

There are basic assignments and settings Intel recommends for your initial compilation. Check the following settings before compiling your design in the Quartus Prime software. Significantly varied compilation results can occur depending on the assignments that you set.

7.2.1 Device Settings

Device assignments determine the timing model that the Quartus Prime software uses during compilation.

Choose the correct speed grade to obtain accurate results and the best optimization. The device size and the package determine the device pin-out and the available resources in the device.

7.2.2 Device Migration Settings

If you anticipate a change to the target device later in the design cycle, either because of changes in your design or other considerations, plan for the change at the beginning of your design cycle.

Whenever you select a target device, you can also list any other compatible devices you can migrate by clicking on the **Migration Devices** button in the **Device** dialog box.

Selecting the migration device and companion device early in the design cycle helps to minimize changes to your design at a later stage.

7.2.3 I/O Assignments

The I/O standards and drive strengths specified for a design affect I/O timing. Specify I/O assignments so that the Quartus Prime software uses accurate I/O timing delays in timing analysis and Fitter optimizations.

If there is no PCB layout requirement, then you do not need to specify pin locations. If your pin locations are not fixed due to PCB layout requirements, then leave the pin locations unconstrained. If your pin locations are already fixed, then make pin assignments to constrain the compilation appropriately.

Use the Assignment Editor and Pin Planner to assign I/O standards and pin locations.

Related Links

- Timing Closure and Optimization on page 116
- Managing Device I/O Pins on page 18

7.2.4 Timing Requirement Settings

For best results, use your real time requirements. If you apply more demanding timing requirements than you need, it may result in increased resource usage, higher power utilization, increased compilation time, or all of these.

Comprehensive timing requirement settings achieve the best results for the following reasons:

- Correct timing assignments enable the software to work hardest to optimize the performance of the timing-critical parts of your design and make trade-offs for performance. This optimization can also save area or power utilization in non-critical parts of your design.
- If enabled, the Quartus Prime software performs physical synthesis optimizations based on timing requirements.

The Quartus Prime TimeQuest Timing Analyzer determines if the design implementation meets the timing requirement. The Compilation Report shows whether your design meets the timing requirements, while the timing analysis reporting commands provide detailed information about the timing paths.

To create timing constraints for the TimeQuest analyzer, create a Synopsys Design Constraints File (`.sdc`). You can also enter constraints in the TimeQuest GUI. Use the `write_sdc` command, or the **Constraints** menu in the TimeQuest analyzer. Click **Write SDC File** to write your constraints to a `.sdc` file. You can add a `.sdc` file to your project on the **Quartus Prime Settings** page under **Timing Analysis Settings**.

If you already have a `.sdc` file in your project, the `write_sdc` command from the command line and the **Write SDC File** option from the TimeQuest GUI allow you to either create a new `.sdc` file that combines the constraints from your current `.sdc` file and any new constraints added through the GUI or command window, or overwrite the existing `.sdc` file with your newly applied constraints.

Ensure that every clock signal has an accurate clock setting constraint. If clocks arrive from a common oscillator, then they are related. Ensure that you set up all related or derived clocks in the constraints correctly. You must constrain all I/O pins that require I/O timing optimization. Specify both minimum and maximum timing constraints as

applicable. If your design contains more than one clock or contains pins with different I/O requirements, make multiple clock settings and individual I/O assignments instead of using a global constraint.

Make any complex timing assignments required in your design, including false path and multicycle path assignments. Common situations for these types of assignments include reset or static control signals (when the time required for a signal to reach a destination is not important) or paths that have more than one clock cycle available for operation in a design. These assignments enable the Quartus Prime software to make appropriate trade-offs between timing paths and can enable the Compiler to improve timing performance in other parts of your design.

Note: To ensure that you apply constraints or assignments to all design nodes, you can report all unconstrained paths in your design with the **Report Unconstrained Paths** command in the **Task** pane of the Quartus Prime TimeQuest Timing Analyzer or the `report_ucp` Tcl command.

Related Links

- Timing Closure and Optimization on page 116
- Advanced Settings (Fitter)
  In Quartus Prime Help
- The Quartus Prime TimeQuest Timing Analyzer
  In *Quartus Prime Pro Edition Handbook Volume 3*
- The Quartus Prime TimeQuest Timing Analyzer
  In *Quartus Prime Standard Edition Handbook Volume 3*
- Quartus Prime TimeQuest Timing Analyzer Cookbook

7.3 Physical Implementation

Most optimization issues involve preserving previous results, reducing area, reducing critical path delay, reducing power consumption, and reducing runtime.

The Quartus Prime software includes advisors to address each of these issues and helps you optimize your design. Run these advisors during physical implementation for advice about your specific design.

You can reduce the time spent on design iterations by following the recommended design practices for designing with Intel devices. Design planning is critical for successful design timing implementation and closure.

Related Links

Design Planning with the Quartus Prime Software
In *Quartus Prime Pro Edition Handbook Volume 1*

7.3.1 Trade-Offs and Limitations

Many optimization goals can conflict with one another, so you might need to resolve conflicting goals.

For example, one major trade-off during physical implementation is between resource usage and critical path timing, because certain techniques (such as logic duplication) can improve timing performance at the cost of increased area. Similarly, a change in

power requirements can result in area and timing trade-offs, such as if you reduce the number of available high-speed tiles, or if you attempt to shorten high-power nets at the expense of critical path nets.

In addition, system cost and time-to-market considerations can affect the choice of device. For example, a device with a higher speed grade or more clock networks can facilitate timing closure at the expense of higher power consumption and system cost.

Finally, not all designs can be realized in a hardware circuit with limited resources and given constraints. If you encounter resource limitations, timing constraints, or power constraints that cannot be resolved by the Fitter, consider rewriting parts of the HDL code.

Related Links

Timing Closure and Optimization on page 116

7.3.2 Reducing Area

By default, the Quartus Prime Fitter might physically spread a design over the entire device to meet the set timing constraints. If you prefer to optimize your design to use the smallest area, you can change this behavior. If you require reduced area, you can enable certain physical synthesis options to modify your netlist to create a more area-efficient implementation, but at the cost of increased runtime and decreased performance.

Related Links

- Netlist Optimizations and Physical Synthesis on page 215
- Reducing Area on page 105
  By default, the Quartus Prime Fitter might physically spread a design over the entire device to meet the set timing constraints.
- Recommended HDL Coding Styles
  In *Quartus Prime Pro Edition Handbook Volume 1*

7.3.3 Reducing Critical Path Delay

To meet complex timing requirements involving multiple clocks, routing resources, and area constraints, the Quartus Prime software offers a close interaction between synthesis, floorplan editing, place-and-route, and timing analysis processes.

By default, the Quartus Prime Fitter tries to meet the specified timing requirements and stops trying when the requirements are met. Therefore, using realistic constraints is important to successfully close timing. If you under-constrain your design, you may get sub-optimal results. By contrast, if you over-constrain your design, the Fitter might over-optimize non-critical paths at the expense of true critical paths. In addition, you might incur an increased area penalty. Compilation time may also increase because of excessively tight constraints.

If your resource usage is very high, the Quartus Prime Fitter might have trouble finding a legal placement. In such circumstances, the Fitter automatically modifies some of its settings to try to trade off performance for area.

The Quartus Prime Fitter offers a number of advanced options that can help you improve the performance of your design when you properly set constraints. Use the Timing Optimization Advisor to determine which options are best suited for your design.

In high-density FPGAs, routing accounts for a major part of critical path timing. Because of this, duplicating or retiming logic can allow the Fitter to reduce delay on critical paths. The Quartus Prime software offers push-button netlist optimizations and physical synthesis options that can improve design performance at the expense of considerable increases of compilation time and area. Turn on only those options that help you keep reasonable compilation times and resource usage. Alternately, you can modify your HDL to manually duplicate or adjust the timing logic.

Related Links

Critical Paths on page 117
Critical paths are timing paths in your design that have a negative slack.

7.3.4 Reducing Power Consumption

The Quartus Prime software has features that help reduce design power consumption. The power optimization options control the power-driven compilation settings for Synthesis and the Fitter.

Related Links

Power Optimization on page 158

7.3.5 Reducing Runtime

Many Fitter settings influence compilation time. Most of the default settings in the Quartus Prime software are set for reduced compilation time. You can modify these settings based on your project requirements.

The Quartus Prime software supports parallel compilation in computers with multiple processors. This can reduce compilation times by up to 15%.

7.4 Using Quartus Prime Tools

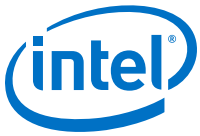
The following sections describe several Quartus Prime tools that you can use to help optimize your design.

7.4.1 Design Analysis

The Quartus Prime software provides tools that help with a visual representation of your design. You can use the RTL Viewer to see a schematic representation of your design before synthesis and place-and-route.The Technology Map Viewer provides a schematic representation of the design implementation in the selected device architecture after synthesis and place-and-route. It can also include timing information.

7.4.2 Advisors

The Quartus Prime software includes several advisors to help you optimize your design and reduce compilation time.



You can complete your design faster by following the recommendations in the advisor. These recommendations are based on your project settings and your design constraints. The advisors are:

- Timing Optimization Advisor
- Power Optimization Advisor
- Compilation Time Advisor

7.4.3 Design Space Explorer II

Use Design Space Explorer II (DSE) to find optimal settings in the Quartus Prime software.

DSE II automatically tries different combinations of netlist optimizations and advanced Quartus Prime software compiler settings, and reports the best settings for your design, based on your chosen primary optimization goal. You can try different seeds with DSE II if you are fairly close to meeting your timing or area requirements and find one seed that meets timing or area requirements. Finally, DSE II can run compilations on a remote compute farm, which shortens the timing closure process.

Related Links

Launch Design Space Explorer Command (Tools Menu)
In Quartus Prime Help

7.5 Document Revision History

Table 33. Document Revision History

Date	Version	Changes
2016.10.31	16.1.0	• Implemented Intel rebranding.
2016.05.03	16.0.0	Removed statements about serial equivalence when using muliple processors.
2015.11.02	15.1.0	Changed instances of *Quartus II* to *Quartus Prime*.
2014.12.15	14.1.0	• Updated location of Fitter Settings, Analysis & Synthesis Settings, and Physical Synthesis Optimizations to Compiler Settings. • Updated DSE II content.
June 2014	14.0.0	Updated format.
November 2013	13.1.0	Minor changes for HardCopy.
May 2013	13.0.0	Added the information about initial compilation requirements. This section was moved from the Area Optimization chapter of the Quartus Prime Handbook. Minor updates to delineate division of Timing and Area optimization chapters.
June 2012	12.0.0	Removed survey link.
November 2011	10.0.3	Template update.
December 2010	10.0.2	Changed to new document template. No change to content.
August 2010	10.0.1	Corrected link
July 2010	10.0.0	Initial release. Chapter based on topics and text in Section III of volume 2.

Related Links

Altera Documentation Archive

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

8 Reducing Compilation Time

You can employ various techniques to reduce to time required for synthesis and fitting in the Quartus Prime Compiler.

8.1 Compilation Time Advisor

A Compilation Time Advisor is available in the Quartus Prime GUI by clicking **Tools ➤ Advisors ➤ Compilation Time Advisor**. This chapter describes all the compilation time optimizing techniques available in the Compilation Time Advisor.

8.2 Strategies to Reduce the Overall Compilation Time

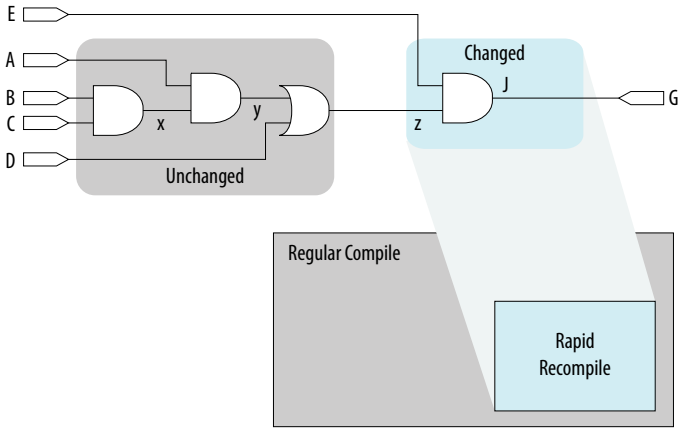
You can use the following strategies to reduce the overall time required to compile your design:

- Parallel compilation (for systems with multiple processor cores)
- Rapid Recompile and Smart Compilation reuse results from a previous compilation to reduce overall compilation time

8.2.1 Running Rapid Recompile

During Rapid Recompile the Compiler reuses previous synthesis and fitting results whenever possible, and does not reprocess unchanged design blocks. Use Rapid Recompile to reduce timing variations and the total recompilation time after making small design changes.

Figure 35. Rapid Recompile

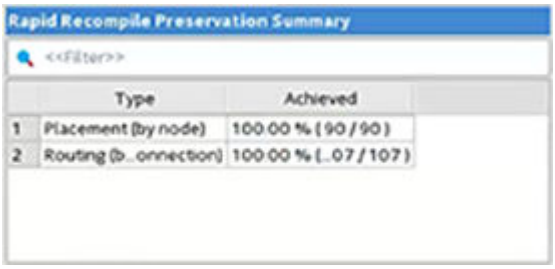


To run Rapid Recompile, follow these steps:

1. Open or create a Quartus Prime project.
2. To start Rapid Recompile following an initial compilation (or after running the Route stage of the Fitter), click **Processing ➤ Start ➤ Start Rapid Recompile**. Rapid Recompile implements the following types of design changes without full recompilation:
   - Changes to nodes tapped by the Signal Tap Logic Analyzer
   - Changes to combinational logic functions
   - Changes to state machine logic (for example, new states, state transition changes)
   - Changes to signal or bus latency or addition of pipeline registers
   - Changes to coefficients of an adder or multiplier
   - Changes register packing behavior of DSP, RAM, or I/O
   - Removal of unnecessary logic
   - Changes to synthesis directives

   The Rapid Recompile Preservation Summary report provides detailed information about the percentage of preserved compilation results.

Figure 36. Rapid Recompile Preservation Summary



8.2.2 Enabling Multi-Processor Compilation

The Compiler can detect and use multiple processors to reduce total compilation time. You can control the number of processors the Compiler uses. The Quartus Prime software can use up to 16 processors to run algorithms in parallel. The Compiler uses parallel compilation by default. To reserve some processors for other tasks, specify a maximum number of processors that the software uses.

You can reduce the compilation time by up to 10% on systems with two processing cores and by up to 20% on systems with four cores. When running timing analysis independently, two processors can reduce the time timing analysis time by an average of 10%. This reduction can reach an average of 15% when using four processors.

The Quartus Prime software does not necessarily use all the processors that you specify during a given compilation. Additionally, the software never uses more than the specified number of processors, enabling you to work on other tasks on your computer without it becoming slow or less responsive. The use of multiple processors does not affect the quality of the fit. For a given Fitter seed and given **Maximum processors allowed** setting on a specific design, the fit is exactly the same and

deterministic, regardless of the target machine and the number of available processors. Different **Maximum processors allowed** specifications produce different results of the same quality. The impact is similar to changing the Fitter seed setting.

To enable multiprocessor compilation, follow these steps:

1. Open or create a Quartus Prime project.
2. To enable multiprocessor compilation, click **Assignments ➤ Settings ➤ Compilation Process Settings**.
3. Under **Parallel compilation**, specify options for the number of processors the Compiler uses.
4. View detailed information about processor in the Parallel Compilation report following compilation.

   To specify the number of processors for compilation at the command line, use the following Tcl command in your script:

```
set_global_assignment -name NUM_PARALLEL_PROCESSORS <value>
```

   In this case, <*value*> is an integer from 1 to 16.

   If you want the Quartus Prime software to detect the number of processors and use all the processors for the compilation, include the following Tcl command in your script:

```
set_global_assignment -name NUM_PARALLEL_PROCESSORS ALL
```

   Note: The Compiler detects Intel Hyper-Threading as a single processor. If your system includes a single processor with Intel Hyper-Threading, set the number of processors to one. Do not use the Intel Hyper-Threading feature for Quartus Prime compilations.

8.3 Reducing Synthesis Time and Synthesis Netlist Optimization Time

You can reduce synthesis time without affecting the Fitter time by reducing your use of netlist optimizations. For tips on reducing synthesis time when using third-party EDA synthesis tools, refer to your synthesis software's documentation.

8.3.1 Settings to Reduce Synthesis Time and Synthesis Netlist Optimization Time

Synthesis netlist and physical synthesis optimization settings can significantly increase the overall compilation time for large designs. Refer to Analysis and Synthesis messages to determine the length of optimization time.

If your design already meets performance requirements without synthesis netlist or physical synthesis optimizations, turn off these options to reduce compilation time. If you require synthesis netlist optimizations to meet performance, optimize partitions of your design hierarchy separately to reduce the overall time spent in Analysis and Synthesis.

8.3.2 Use Appropriate Coding Style to Reduce Synthesis Time

Your HDL coding style can also affect the synthesis time. For example, if you want to infer RAM blocks from your code, you must follow the guidelines for inferring RAMs. If RAM blocks are not inferred properly, the software implements those blocks as registers.

If you are trying to infer a large memory block, the software consumes more resources in the FPGA. This can cause routing congestion and increasing compilation time significantly. If you see high routing utilizations in certain blocks, it is a good idea to review the code for such blocks.

Related Links

Recommended HDL Coding Styles
In *Quartus Prime Pro Edition Handbook Volume 1: Design and Synthesis*

8.4 Reducing Placement Time

The time required to place a design depends on two factors: the number of ways the logic in your design can be placed in the device, and the settings that control the amount of effort required to find a good placement.

You can reduce the placement time by changing the settings for the placement algorithm.

Sometimes there is a trade-off between placement time and routing time. Routing time can increase if the placer does not run long enough to find a good placement. When you reduce placement time, ensure that it does not increase routing time and negate the overall time reduction.

8.4.1 Placement Effort Multiplier Settings

You can control the amount of time the Fitter spends in placement by reducing with the **Placement Effort Multiplier** option.

Click **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Fitter)** and specify a value for Placement Effort Multiplier. The default is `1.0`. Legal values must be greater than 0 and can be non-integer values. Numbers between 0 and 1 can reduce fitting time, but also can reduce placement quality and design performance.

8.5 Reducing Routing Time

The routing time is usually not a significant amount of the compilation time.The time required to route a design depends on three factors: the device architecture, the placement of your design in the device, and the connectivity between different parts of your design.

If your design requires a long time to route, perform one or more of the following actions:

- Check for routing congestion.
- Turn off **Fitter Aggressive Routability Optimization**.

8.5.1 Identifying Routing Congestion with the Chip Planner

To identify areas of routing congestion in your design:

1. Click **Tools ➤ Chip Planner**.
2. To view the routing congestion in the Chip Planner, double-click the **Report Routing Utilization** command in the **Tasks** list.
3. Click **Preview** in the **Report Routing Utilization** dialog box to preview the default congestion display.
4. Change the **Routing utilization type** to display congestion for specific resources. The default display uses dark blue for 0% congestion and red for 100%.
5. Adjust the slider for **Threshold percentage** to change the congestion threshold level.

The Quartus Prime compilation messages contain information about average and peak interconnect usage. Peak interconnect usage over 75%, or average interconnect usage over 60%, could be an indication that it might be difficult to fit your design. Similarly, peak interconnect usage over 90%, or average interconnect usage over 75%, are likely to have increased chances of not getting a valid fit.

8.5.1.1 Areas with Routing Congestion

Even if average congestion is not very high, your design may have areas where congestion is very high in a specific type of routing. You can use the Chip Planner to identify areas of high congestion for specific interconnect types.

- You can change the connections in your design to reduce routing congestion
- If the area with routing congestion is in a LogicLock Plus region or between LogicLock Plus regions, change or remove the LogicLock Plus regions and recompile your design.
  - If the routing time remains the same, the time is a characteristic of your design and the placement
  - If the routing time decreases, consider changing the size, location, or contents of LogicLock Plus regions to reduce congestion and decrease routing time.

Related Links

Analyzing and Optimizing the Design Floorplan on page 190
This chapter discusses how the Chip Planner and LogicLock Plus regions help you improve your design's floorplan.

8.5.1.2 Congestion due to HDL Coding style

Sometimes, routing congestion may be a result of the HDL coding style used in your design. After you identity congested areas using the Chip Planner, review the HDL code for the blocks placed in those areas to determine whether you can reduce interconnect usage by code changes.

Related Links

Recommended HDL Coding Styles
In *Quartus Prime Pro Edition Handbook Volume 1*

8.6 Reducing Static Timing Analysis Time

If you are performing timing-driven synthesis, the Quartus Prime software runs the TimeQuest analyzer during Analysis and Synthesis.

The Quartus Prime Fitter also runs the TimeQuest analyzer during placement and routing. If there are incorrect constraints in the Synopsys Design Constraints File (`.sdc`), the Quartus Prime software may spend unnecessary time processing constraints several times.

- If you do not specify false paths and multicycle paths in your design, the TimeQuest analyzer may analyze paths that are not relevant to your design.
- If you redefine constraints in the `.sdc` files, the TimeQuest analyzer may spend additional time processing them. To avoid this situation, look for indications that Synopsis design constraints are being redefined in the compilation messages, and update the `.sdc` file.
- Ensure that you provide the correct timing constraints to your design, because the software cannot assume design intent, such as which paths to consider as false paths or multicycle paths. When you specify these assignments correctly, the TimeQuest analyzer skips analysis for those paths, and the Fitter does not spend additional time optimizing those paths.

8.7 Setting Process Priority

It might be necessary to reduce the computing resources allocated to the compilation at the expense of increased compilation time. It can be convenient to reduce the resource allocation to the compilation with single processor machines if you must run other tasks at the same time.

Related Links

Processing Page (Options Dialog Box)
In Quartus Prime Help.

8.8 Document Revision History

Table 34. Document Revision History

Date	Version	Changes
2017.05.08	17.0.0	• Clarified impact of multiprocessor compilation on fit quality. • Removed reference to deprecated Fitter Effort Logic Option. • Removed section: *Preserving Routing with Incremental Compilation*.
2016.10.31	16.1.0	• Implemented Intel rebranding.
2016.05.02	16.0.0	• Corrected typo in Using Parallel Compilation with Multiple Processors. • Removed information about deprecated physical synthesis options.
2015.11.02	15.1.0	Changed instances of *Quartus II* to *Quartus Prime*.
2014.12.15	14.1.0	• Updated location of Fitter Settings, Analysis & Synthesis Settings, and Physical Synthesis Optimizations to Compiler Settings. • Added information about Rapid Recompile feature.
2014.08.18	14.0a10.0	Added restriction about smart compilation in Arria 10 devices.
June 2014	14.0.0	Updated format.

continued...

Date	Version	Changes
May 2013	13.0.0	Removed the "Limit to One Fitting Attempt", "Using Early Timing Estimation", "Final Placement Optimizations", and "Using Rapid Recompile" sections. Updated "Placement Effort Multiplier Settings" section. Updated "Identifying Routing Congestion in the Chip Planner" section. General editorial changes throughout the chapter.
June 2012	12.0.0	Removed survey link.
November 2011	11.0.1	Template update.
May 2011	11.0.0	• Updated "Using Parallel Compilation with Multiple Processors". • Updated "Identifying Routing Congestion in the Chip Planner". • General editorial changes throughout the chapter.
December 2010	10.1.0	• Template update. • Added details about peak and average interconnect usage. • Added new section "Reducing Static Timing Analysis Time". • Minor changes throughout chapter.
July 2010	10.0.0	Initial release.

Related Links

Altera Documentation Archive

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

9 Timing Closure and Optimization

9.1 About Timing Closure and Optimization

This chapter describes techniques to improve timing performance when designing for Intel devices.

The application techniques vary between designs. Applying each technique does not always improve results. Settings and options in the Quartus Prime software have default values that provide the best trade-off between compilation time, resource utilization, and timing performance. You can adjust these settings to determine whether other settings provide better results for your design.

9.2 Optimize Multi-Corner Timing

Due to process variations and changes in operating conditions, delays on some paths can be significantly smaller than those in the slow corner timing model. This can result in hold time violations on those paths, and in rare cases, additional setup time violations.

Also, because of the small process geometries of newer device families, the slowest circuit performance of designs targeting these devices does not necessarily occur at the highest operating temperature. The temperature at which the circuit is slowest depends on the selected device, the design, and the compilation results. Therefore, the Quartus Prime software provides newer device families with three different timing corners—Slow 85°C corner, Slow 0°C corner, and Fast 0°C corner. For other device families, two timing corners are available—Fast 0°C and Slow 85°C corner.

The **Optimize multi-corner timing** option directs the Fitter to consider all corner timing delays, including both fast-corner timing and slow-corner timing, during optimization to meet timing requirements at all process corners and operating conditions. By default, this option is on, and the Fitter optimizes designs considering multi-corner delays in addition to slow-corner delays, for example, from the fast-corner timing model, which is based on the fastest manufactured device, operating under high-voltage conditions

The **Optimize multi-corner timing** option helps to create a design implementation that is more robust across process, temperature, and voltage variations. Turning on this option increases compilation time by approximately 10%.

When this option is off, the Fitter optimizes designs considering only slow-corner delays from the slow-corner timing model (slowest manufactured device for a given speed grade, operating in low-voltage conditions).

9.3 Critical Paths

Critical paths are timing paths in your design that have a negative slack. These timing paths can span from device I/Os to internal registers, registers to registers, or from registers to device I/Os.

The slack of a path determines its criticality; slack appears in the timing analysis report, which you can generate using the TimeQuest Timing Analyzer.

Design analysis for timing closure is a fundamental requirement for optimal performance in highly complex designs.The analytical capability of the Chip Planner helps you close timing on complex designs.

Related Links

- Reducing Critical Path Delay on page 105
  To meet complex timing requirements involving multiple clocks, routing resources, and area constraints, the Quartus Prime software offers a close interaction between synthesis, floorplan editing, place-and-route, and timing analysis processes.
- Displaying Path Reports with the TimeQuest Timing Analyzer on page 120
  The TimeQuest timing analyzer generate reports with information about all valid register-to-register paths.

9.3.1 Viewing Critical Paths

Viewing critical paths in the Chip Planner shows why a specific path is failing. You can see if any modification in the placement can reduce the negative slack. To display paths in the floorplan, perform a timing analysis and display results on the TimeQuest Timing Analyzer.

9.4 Initial Compilation: Optional Fitter Settings

The Fitter offers many optional settings; however, this section focuses only on the optional timing-optimization related Fitter settings, which are the **Optimize Hold Timing**, **Optimize Multi-Corner Timing,** and **Fitter Aggressive Routability Optimization**.

Caution: The settings required to optimize different designs could be different. The group of settings that work best for one design may not produce the best result for another design.

Related Links

Advanced Fitter Setting Dialog Box
In Quartus Prime Help

9.4.1 Optimize Hold Timing

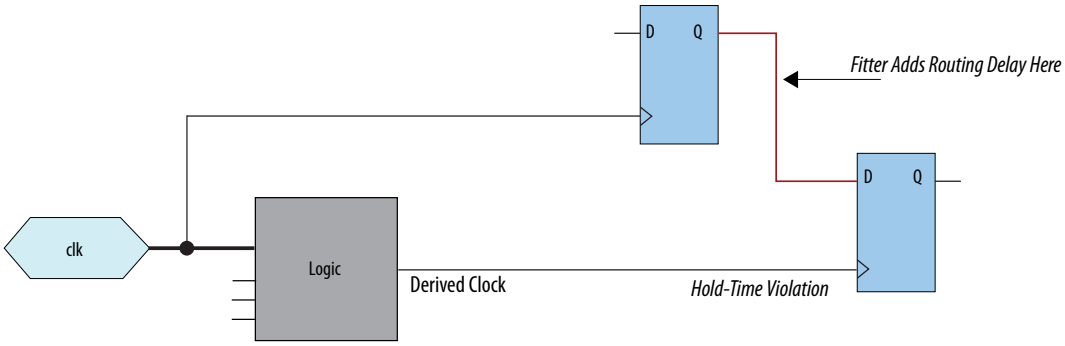
The **Optimize Hold Timing** option directs the Quartus Prime software to optimize minimum delay timing constraints. By default, the Quartus Prime software optimizes hold timing for all paths for designs for supported devices. By default, the Quartus Prime software optimizes hold timing only for I/O paths and minimum t_{PD} paths for older devices.

When you turn on **Optimize Hold Timing** in the **Advanced Fitter Settings** dialog box, the Quartus Prime software adds delay to paths to ensure that your design meets the minimum delay requirements. If you select **I/O Paths and Minimum TPD Paths**, the Fitter works to meet the following criteria:

- Hold times (t_H) from the device input pins to the registers
- Minimum delays from I/O pins to I/O registers or from I/O registers to I/O pins
- Minimum clock-to-out time (t_{CO}) from registers to output pins

If you select **All Paths**, the Fitter also works to meet hold requirements from registers to registers, as highlighted in blue in the figure, in which a derived clock generated with logic causes a hold time problem on another register.

Figure 37. Optimize Hold Timing Option Fixing an Internal Hold Time Violation



However, if your design still has internal hold time violations between registers, correct the violations by manually adding some delays by instantiating LCELL primitives, or by making changes to your design, such as using a clock enable signal instead of a derived or gated clock.

Related Links

Recommended Design Practices
In *Quartus Prime Pro Edition Handbook Volume 1*

9.4.2 Fitter Aggressive Routability Optimization

The **Fitter Aggressive Routability Optimizations** logic option allows you to specify whether the Fitter aggressively optimizes for routability. Performing aggressive routability optimizations may decrease design speed, but may also reduce routing wire usage and routing time.

This option is useful if routing resources are resulting in no-fit errors, and you want to reduce routing wire use.

The table lists the settings for the **Fitter Aggressive Routability Optimizations** logic option.

Table 35. Fitter Aggressive Routability Optimizations Logic Option Settings

Settings	Description
Always	The Fitter always performs aggressive routability optimizations. If you set the Fitter Aggressive Routability Optimizations logic option to Always, reducing wire utilization may affect the performance of your design.
Never	The Fitter never performs aggressive routability optimizations. If improving timing is more important than reducing wire usage, then set this option to Automatically or Never.
Automatically	The Fitter performs aggressive routability optimizations automatically, based on the routability and timing requirements of the design. If improving timing is more important than reducing wire usage, then set this option to Automatically or Never.

9.5 Design Analysis

The initial compilation establishes whether the design achieves a successful fit and meets the specified timing requirements. This section describes how to analyze your design results in the Quartus Prime software.

9.5.1 Ignored Timing Constraints

The Quartus Prime software ignores illegal, obsolete, and conflicting constraints.

You can view a list of ignored constraints in the TimeQuest GUI by clicking **Reports ➤ Report Ignored Constraints** or by typing the following command to generate a list of ignored timing constraints:

```
report_sdc -ignored -panel_name "Ignored Constraints"
```

Analyze any constraints that the Quartus Prime software ignores. If necessary, correct the constraints and recompile your design before proceeding with design optimization.

You can view a list of ignored assignment in the **Ignored Assignment Report** generated by the Fitter.

Related Links

- Report Ignored Constraints Command
  In Quartus Prime Help
- Fitter Summary Reports
  In Quartus Prime Help

9.5.2 I/O Timing

TimeQuest analyzer supports the Synopsys Design Constraints (SDC) format for constraining your design. When using the TimeQuest analyzer for timing analysis, use the `set_input_delay` constraint to specify the data arrival time at an input port with respect to a given clock. For output ports, use the `set_output_delay` command to specify the data arrival time at an output port's receiver with respect to a given clock. You can use the `report_timing` Tcl command to generate the I/O timing reports.

The I/O paths that do not meet the required timing performance are reported as having negative slack and are highlighted in red in the TimeQuest analyzer **Report** pane. In cases where you do not apply an explicit I/O timing constraint to an I/O pin,

the Quartus Prime timing analysis software still reports the **Actual** number, which is the timing number that must be met for that timing parameter when the device runs in your system.

Related Links

Creating I/O Requirements
In *Quartus Prime Pro Edition Handbook Volume 3*

9.5.3 Register-to-Register Timing Analysis

Your design meets timing requirements when you do not have negative slack on any register-to-register path on any of the clock domains. When timing requirements are not met, a report on the failed paths can uncover more detail.

9.5.3.1 Displaying Path Reports with the TimeQuest Timing Analyzer

The TimeQuest timing analyzer generate reports with information about all valid register-to-register paths. To view all timing summaries, run the **Report All Summaries** command by double-clicking **Report All Summaries** in the **Tasks** pane.

If any clock domains have failing paths (highlighted in red in the **Report** pane), right-click the clock name listed in the **Clocks Summary** pane and select **Report Timing** to get more details.

When you select a path in the **Summary of Paths** tab, the path detail pane displays all the path information. The **Extra Fitter Information** tab offers visual representation of the path location on the physical device. This can reveal whether the timing failure is distance related, due to the source and destination node being too close or too far.

The **Data Path** tab displays the Data Arrival Path and the Data Required Path. Using the incremental information you can determine the path segments contributing the most to the timing violations. The **Waveform** tab shows the signals in the time domain, and plots the slack between arrival data and required data.

To assess which areas in your design may benefit from reducing the number of logic levels, you can use the RTL Viewer or Technology Map Viewer to see schematic (gate-level or technology-mapped) representations of your design netlist. To locate a timing path in one of the viewers, right-click a path in the timing report, point to **Locate**, and select either **Locate in RTL Viewer** or **Locate in Technology Map Viewer**. The Chip Planner can also be used to investigate the physical layout of a path in more detail.

Related Links

- Generating Timing Reports
  In *Quartus Prime Pro Edition Handbook Volume 3*
- When to Use the Netlist Viewers: Analyzing Design Problems
  In *Quartus Prime Pro Edition Handbook Volume 1*

9.5.3.2 Tips for Analyzing Failing Paths

When you are analyzing failing paths, examine the reports and waveforms to determine if the correct constraints are being applied, and add timing exceptions as appropriate. A multicycle constraint relaxes setup or hold relationships by the specified

number of clock cycles. A false path constraint specifies paths that can be ignored during timing analysis. Both constraints allow the Fitter to work harder on affected paths.

Focus on improving the paths that show the worst slack. The Fitter works hardest on paths with the worst slack. If you fix these paths, the Fitter might be able to improve the other failing timing paths in the design.

Check for particular nodes that appear in many failing paths. These nodes appear at the top of the list in a timing report panel, along with their minimum slacks. Look for paths that have common source registers, destination registers, or common intermediate combinational nodes. In some cases, the registers might not be identical, but are part of the same bus.

In the timing analysis report panels, click the **From** or **To** column headings to sort the paths by source or destination registers. If you see common nodes, these nodes indicate areas of your design that might be improved through source code changes or Quartus Prime optimization settings. Constraining the placement for just one of the paths might decrease the timing performance for other paths by moving the common node further away in the device.

Related Links

- Exploring Paths in the Chip Planner on page 196
  Use the Chip Planner to explore paths between logic elements. The following examples use the Chip Planner to traverse paths from the Timing Analysis report.
- Design Evaluation for Timing Closure on page 141
  Follow the guidelines in this section when you encounter timing failures in a design.

9.5.3.3 Tips for Analyzing Failing Clock Paths that Cross Clock Domains

When analyzing clock path failures, check whether these paths cross two clock domains. This is the case if the **From Clock** and **To Clock** in the timing analysis report are different.

Figure 38. Different Value in From Clock and To Clock Field

Setup Transfers

	From Clock	To Clock	RR Paths	FR Paths	RF Paths	FF Paths
1	clkin	clkin	21	0	0	0
2	clkin	clkout	false path	0	0	0
3	clkout	clkout	31	0	0	0

There can also be paths that involve a different clock in the middle of the path, even if the source and destination register clock are the same.

When you run `report_timing` on your design, the report shows the launch clock and latch clock for each failing path. Check whether these failing paths between these clock domains should be analyzed synchronously. If the failing paths are not to be analyzed synchronously, they must be set as false paths. Also check the relationship between the launch clock and latch clock to make sure it is realistic and what you expect from your knowledge of the design. For example, the path can start at a rising edge and end at a falling edge, which reduces the setup relationship by one half clock cycle.

Review the clock skew reported in the Timing Report. A large skew may indicate a problem in your design, such as a gated clock or a problem in the physical layout (for example, a clock using local routing instead of dedicated clock routing). When you have made sure the paths are analyzed synchronously and that there is no large skew on the path, and that the constraints are correct, you can analyze the data path. These steps help you fine tune your constraints for paths across clock domains to ensure you get an accurate timing report.

Check if the PLL phase shift is reducing the setup requirement. You might be able to adjust this using PLL parameters and settings.

Paths that cross clock domains are generally protected with synchronization logic (for example, FIFOs or double-data synchronization registers) to allow asynchronous interaction between the two clock domains. In such cases, you can ignore the timing paths between registers in the two clock domains while running timing analysis, even if the clocks are related.

The Fitter attempts to optimize all failing timing paths. If there are paths that can be ignored for optimization and timing analysis, but the paths do not have constraints that instruct the Fitter to ignore them, the Fitter tries to optimize those paths as well. In some cases, optimizing unnecessary paths can prevent the Fitter from meeting the timing requirements on timing paths that are critical to the design. It is beneficial to specify all paths that can be ignored by setting false path constraints on them, so that the Fitter can put more effort into the paths that must meet their timing requirements instead of optimizing paths that can be ignored.

Related Links

report_clock_transfers (::quartus::sta)
In Quartus Prime Help

9.5.3.4 Tips for Analyzing Paths from/to the Source and Destination of Critical Path

When analyzing the failing paths in a design, it is often helpful to get a fuller picture of the many interactions the fitter may be working on around the paths.

To understand what may be pulling on a critical path, the following `report_timing` command can be useful.

1. In the project directory, run the `report_timing` command to find the nodes in a critical path.
2. Copy the code below in a `.tcl` file, and replace the first two variable with the node names from the **From Node** and **To Node** columns of the worst path. The script analyzes the path between the worst source and destination registers.

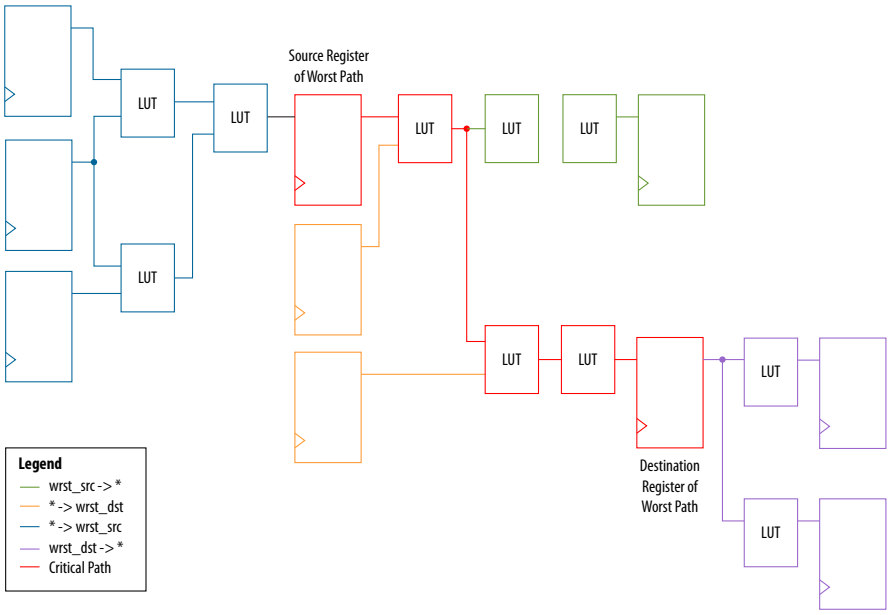
```
set wrst_src <insert_source_of_worst_path_here>
set wrst_dst <insert_destination_of_worst_path_here>
report_timing -setup -npaths 50 -detail path_only -from $wrst_src \
-panel_name "Worst Path||wrst_src -> *"
report_timing -setup -npaths 50 -detail path_only -to $wrst_dst \
-panel_name "Worst Path||* -> wrst_dst"
report_timing -setup -npaths 50 -detail path_only -to $wrst_src \
```

```
-panel_name "Worst Path||* -> wrst_src"
report_timing -setup -npaths 50 -detail path_only -from $wrst_dst \
-panel_name "Worst Path||wrst_dst -> *"
```

3. From the **Script** menu, source the `.tcl` file.
4. In the resulting timing panel, locate timing failed paths (highlighted in red) in the Chip Planner, and view information such as distance between the nodes and large fanouts.

   The figure shows a simplified example of what these reports analyzed.

Figure 39. Timing Report



The critical path of the design is in red. The relation between the `.tcl` script and the figure is:

- The first two lines show everything inside the two endpoints of the critical path that are pulling them in different directions.
  - The first `report_timing` command analyzes all paths the source is driving, shown in green.
  - The second `report_timing` command analyzes all paths going to the destination, including the critical path, shown in orange.
- The last two `report_timing` commands show everything outside of the endpoints pulling them in other directions.

If any of these neighboring paths have slacks near the critical path, the Fitter is balancing these paths with the critical path, trying to achieve the best slack.

9.5.3.5 Tips for Creating a .tcl Script to Monitor Critical Paths Across Compiles

Many designs have the same critical paths show up after each compile, but some suffer from having critical paths bounce around between different hierarchies, changing with each compile.

This could happen in high speed designs where many register to register paths have very little slack. Different placements can then result in timing failures in the marginal paths. In designs like this, create a `TQ_critical_paths.tcl` script in the project directory. For a given compile, view the critical paths and then write a generic `report_timing` command to capture those paths. For example, if several paths fail in a low-level hierarchy, you can add the following command:

```
report_timing –setup –npaths 50 –detail path_only \
–to “main_system: main_system_inst|app_cpu:cpu|*” \
–panel_name “Critical Paths||s: * -> app_cpu”
```

If there is a specific path, such as a bit of a state-machine going to other `*count_sync*` registers, you can add a command as shown by the following:

```
report_timing –setup –npaths 50 –detail path_only \
–from “main_system: main_system_inst|egress_count_sm:egress_inst|update” \
–to “*count_sync*” –panel_name “Critical Paths||s: egress_sm|update -> count_sync”
```

This file can be sourced in the TimeQuest timing analyzer after every compilation, and new `report_timing` commands can be added as new critical paths appear. This helps you monitor paths that consistently fail and paths that are only marginal, so you can prioritize effectively.

9.5.3.6 Global Routing Resources

Global routing resources are designed to distribute high fan-out, low-skew signals (such as clocks) without consuming regular routing resources. Depending on the device, these resources can span the entire chip, or some smaller portion, such as a quadrant. The Quartus Prime software attempts to assign signals to global routing resources automatically, but you might be able to make more suitable assignments manually.

For details about the number and types of global routing resources available, refer to the relevant device handbook.

Check the global signal utilization in your design to ensure that the appropriate signals have been placed on the global routing resources. In the Compilation Report, open the Fitter report and click **Resource Section**. Analyze the Global & Other Fast Signals and Non-Global High Fan-out Signals reports to determine whether any changes are required.

You might be able to reduce skew for high fan-out signals by placing them on global routing resources. Conversely, you can reduce the insertion delay of low fan-out signals by removing them from global routing resources. Doing so can improve clock enable timing and control signal recovery/removal timing, but increases clock skew. Use the **Global Signal** setting in the Assignment Editor to control global routing resources.

9.6 Timing Optimization

Use the following guidelines if your design does not meet its timing requirements.

9.6.1 Displaying Timing Closure Recommendations for Failing Paths

Use the **Timing Closure Recommendations** report to get specific recommendations about failing paths in your design and changes that you can make to potentially fix the failing paths. The **Report Timing Closure Recommendations** task is available in the Custom Reports section of the **Tasks** pane of the TimeQuest analyzer.

1. Select the **Report Timing Closure Recommendations** task to open the **Report Timing Closure Recommendations** dialog box.
2. Select paths based on the clock domain, filter by nodes on path, and choose the number of paths to analyze.
3. After running the **Report Timing Closure Recommendations** task in the TimeQuest analyzer, examine the reports in the **Report Timing Closure Recommendations** folder in the **Report** pane of the TimeQuest analyzer GUI. Each recommendation has star symbols (*) associated with it. Recommendations with more stars are more likely to help you close timing on your design.

The reports give you the most probable causes of failure for each path being analyzed and show recommendations that may help you fix the failing paths.

The reports are organized into sections, depending on the type of issues found in the design, such as large clock skew, restricted optimizations, unbalanced logic, skipped optimizations, coding style that has too many levels of logic between registers, or region or partition constraints specific to your project.

For detailed analysis of the critical paths, run the `report_timing` command on specified paths. In the **Extra Fitter Information** tab of the **Path** report panel, you will also see detailed fitter-related information that may help you visualize the issue.

Related Links

Report Timing Closure Recommendations Dialog Box
In Quartus Prime Help

9.6.2 Timing Optimization Advisor

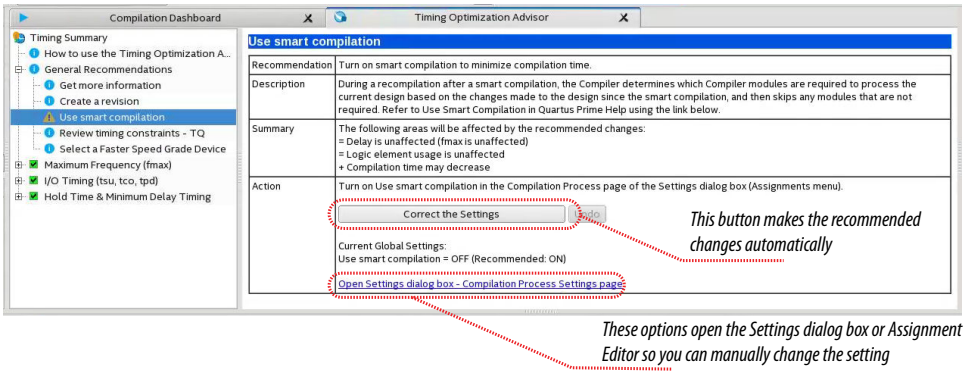
While the TimeQuest **Report Timing Closure Recommendations** task gives specific recommendations to fix failing paths, the Timing Optimization Advisor gives more general recommendations to improve timing performance for a design.

The Timing Optimization Advisor guides you in making settings that optimize your design to meet your timing requirements. To run the Timing Optimization Advisor click **Tools ➤ Advisors ➤ Timing Optimization Advisor**. This advisor describes many of the suggestions made in this section.

When you open the Timing Optimization Advisor after compilation, you can find recommendations to improve the timing performance of your design. Some of the suggestions in these advisors can contradict each other. Intel recommends evaluating these options and choosing the settings that best suit the given requirements.

The example shows the Timing Optimization Advisor after compiling a design that meets its frequency requirements, but requires setting changes to improve the timing.

Figure 40. Timing Optimization Advisor



When you expand one of the categories in the Timing Optimization Advisor, such as **Maximum Frequency (fmax)** or **I/O Timing (tsu, tco, tpd)**, the recommendations are divided into stages. The stages show the order in which to apply the recommended settings.

The first stage contains the options that are easiest to change, make the least drastic changes to your design optimization, and have the least effect on compilation time.

Icons indicate whether each recommended setting has been made in the current project. In the figure, the checkmark icons in the list of recommendations for Stage 1 indicate recommendations that are already implemented. The warning icons indicate recommendations that are not followed for this compilation. The information icons indicate general suggestions. For these entries, the advisor does not report whether these recommendations were followed, but instead explains how you can achieve better performance. For a legend that provides more information for each icon, refer to the "How to use" page in the Timing Optimization Advisor.

Each recommendation provides a link to the appropriate location in the Quartus Prime GUI where you can change the settings. For example, consider the **Synthesis Netlist Optimizations** page of the **Settings** dialog box or the **Global Signals category** in the Assignment Editor. This approach provides the most control over which settings are made and helps you learn about the settings in the software. In some cases, you can also use the **Correct the Settings** button to automatically make the suggested change to global settings.

For some entries in the Timing Optimization Advisor, a button appears that allows you to further analyze your design and gives you more information. The advisor provides a table with the clocks in the design and indicates whether they have been assigned a timing constraint.

9.6.3 I/O Timing Optimization

This stage of design optimization focuses on I/O timing. Ensure that you have made the appropriate assignments described in the *Initial Compilation: Required Settings* section of the *Design Optimization Overview* chapter. Also ensure that resource utilization is satisfactory before proceeding with I/O timing optimization. The suggestions provided in this section are applicable to all Intel FPGA families and to the family of CPLDs.

Because changes to the I/O paths affect the internal register-to-register timing, complete this stage before proceeding to the register-to-register timing optimization stage as described in *Register-to-Register Timing Optimization Techniques (LUT-Based Devices)*.

The options presented in this section address how to improve I/O timing, including the setup delay (t_{SU}), hold time (t_H), and clock-to-output (t_{CO}) parameters.

Improving Setup and Clock-to-Output Times Summary on page 127
The table lists the recommended order in which to use techniques to reduce t_{SU} and t_{CO} times.

Timing-Driven Compilation on page 128
This option moves registers into I/O elements if required to meet t_{SU} or t_{CO} assignments, duplicating the register if necessary (as in the case in which a register fans out to multiple output locations).

Fast Input, Output, and Output Enable Registers on page 128
You can place individual registers in I/O cells manually by making fast I/O assignments with the Assignment Editor.

Programmable Delays on page 129
You can use various programmable delay options to minimize the t_{SU} and t_{CO} times.

Use PLLs to Shift Clock Edges on page 130
Using a PLL typically improves I/O timing automatically.

Use Fast Regional Clock Networks and Regional Clocks Networks on page 130
Regional clocks provide the lowest clock delay and skew for logic contained in a single quadrant.

Spine Clock Limitations on page 130
If your project has high clock routing demands, due to limitations in the Quartus Prime software, you may see spine clock errors.

9.6.3.1 Improving Setup and Clock-to-Output Times Summary

The table lists the recommended order in which to use techniques to reduce t_{SU} and t_{CO} times. "Yes" indicates which timing parameters are affected by each technique. Reducing t_{SU} times increases hold (t_H) times.

Table 36. Improving Setup and Clock-to-Output Times

Technique	Affects t_{SU}	Affects t_{CO}
Ensure that the appropriate constraints are set for the failing I/Os (refer to *Initial Compilation: Required Settings*)	Yes	Yes
Use timing-driven compilation for I/O (refer to *Fast Input, Output, and Output Enable Registers*)	Yes	Yes
Use fast input register (refer to *Programmable Delays*)	Yes	N/A
Use fast output register, fast output enable register, and fast OCT register (refer to *Programmable Delays*)	N/A	Yes
Decrease the value of **Input Delay from Pin to Input Register** or set **Decrease Input Delay to Input Register = ON**	Yes	N/A
Decrease the value of **Input Delay from Pin to Internal Cells** or set **Decrease Input Delay to Internal Cells = ON**	Yes	N/A

continued...

Technique	Affects t_{SU}	Affects t_{CO}
Decrease the value of **Delay from Output Register to Output Pin** or set **Increase Delay to Output Pin = OFF** (refer to *Fast Input, Output, and Output Enable Registers*)	N/A	Yes
Increase the value of **Input Delay from Dual-Purpose Clock Pin to Fan-Out Destinations** (refer to *Fast Input, Output, and Output Enable Registers*)	Yes	N/A
Use PLLs to shift clock edges	Yes	Yes
Increase the value of **Delay to output enable pin** or set **Increase delay to output enable pin** (refer to *Use PLLs to Shift Clock Edges*)	N/A	Yes
Note to table : 1. These options may not apply to all device families.		

Related Links

Initial Compilation: Required Settings on page 102
There are basic assignments and settings Intel recommends for your initial compilation. Check the following settings before compiling your design in the Quartus Prime software.

9.6.3.2 Timing-Driven Compilation

This option moves registers into I/O elements if required to meet t_{SU} or t_{CO} assignments, duplicating the register if necessary (as in the case in which a register fans out to multiple output locations). This option is turned on by default and is a global setting.

The **Optimize IOC Register Placement for Timing** option affects only pins that have a t_{SU} or t_{CO} requirement. Using the I/O register is possible only if the register directly feeds a pin or is fed directly by a pin. This setting does not affect registers with any of the following characteristics:

- Have combinational logic between the register and the pin
- Are part of a carry or cascade chain
- Have an overriding location assignment
- Use the asynchronous load port and the value is not 1 (in device families where the port is available)

Registers with the characteristics listed are optimized using the regular Quartus Prime Fitter optimizations.

Related Links

Optimize IOC Register Placement for Timing Logic Option
In Quartus Prime Help

9.6.3.3 Fast Input, Output, and Output Enable Registers

You can place individual registers in I/O cells manually by making fast I/O assignments with the Assignment Editor.By default, with correct timing assignments, the Fitter places the I/O registers in the correct I/O cell or in the core, to meet the performance requirement.

If the fast I/O setting is on, the register is always placed in the I/O element. If the fast I/O setting is off, the register is never placed in the I/O element. This is true even if the **Optimize IOC Register Placement for Timing** option is turned on. If there is no fast I/O assignment, the Quartus Prime software determines whether to place registers in I/O elements if the **Optimize IOC Register Placement for Timing** option is turned on.

You can also use the four fast I/O options (**Fast Input Register**, **Fast Output Register**, **Fast Output Enable Register**, and **Fast OCT Register**) to override the location of a register that is in a LogicLock Plus region and force it into an I/O cell. If you apply this assignment to a register that feeds multiple pins, the register is duplicated and placed in all relevant I/O elements.

For more information about the **Fast Input Register** option, **Fast Output Register** option, **Fast Output Enable Register** option, and **Fast OCT (on-chip termination) Register** option, refer to Quartus Prime Help.

Related Links

- Fast Input Register logic option
- Fast Output Register logic option
- Fast Output Enable Register logic option
- Fast OCT Register logic option

9.6.3.4 Programmable Delays

You can use various programmable delay options to minimize the t_{SU} and t_{CO} times. Programmable delays are advanced options to use only after you compile a project, check the I/O timing, and determine that the timing is unsatisfactory.

For Arria, Cyclone, MAX® II, MAX V, and Stratix series devices, the Quartus Prime software automatically adjusts the applicable programmable delays to help meet timing requirements. For detailed information about the effect of these options, refer to the device family handbook or data sheet.

After you have made a programmable delay assignment and compiled the design, you can view the implemented delay values for every delay chain and every I/O pin in the **Delay Chain Summary** section of the Compilation Report.

You can assign programmable delay options to supported nodes with the Assignment Editor. You can also view and modify the delay chain setting for the target device with the Chip Planner and Resource Property Editor. When you use the Resource Property Editor to make changes after performing a full compilation, recompiling the entire design is not necessary; you can save changes directly to the netlist. Because these changes are made directly to the netlist, the changes are not made again automatically when you recompile the design. The change management features allow you to reapply the changes on subsequent compilations.

Although the programmable delays in newer devices are user-controllable, Intel recommends their use for advanced users only. However, the Quartus Prime software might use the programmable delays internally during the Fitter phase.

For details about the programmable delay logic options available for Intel devices, refer to the following Quartus Prime Help topics:

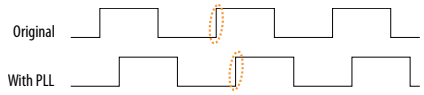
Related Links

- Input Delay from Pin to Input Register logic option
- Input Delay from Pin to Internal Cells logic option
- Output Enable Pin Delay logic option
- Delay from Output Register to Output Pin logic option
- Input Delay from Dual-Purpose Clock Pin to Fan-Out Destinations logic option

9.6.3.5 Use PLLs to Shift Clock Edges

Using a PLL typically improves I/O timing automatically. If the timing requirements are still not met, most devices allow the PLL output to be phase shifted to change the I/O timing. Shifting the clock backwards gives a better t_H at the expense of t_{SU}, while shifting it forward gives a better t_{SU} at the expense of t_H. You can use this technique only in devices that offer PLLs with the phase shift option.

Figure 41. Shift Clock Edges Forward to Improve t_{SU} at the Expense of t_H



You can achieve the same type of effect in certain devices by using the programmable delay called **Input Delay from Dual Purpose Clock Pin to Fan-Out Destinations**.

9.6.3.6 Use Fast Regional Clock Networks and Regional Clocks Networks

Regional clocks provide the lowest clock delay and skew for logic contained in a single quadrant.In general, fast regional clocks have less delay to I/O elements than regional and global clocks, and are used for high fan-out control signals. Placing clocks on these low-skew and low-delay clock nets provides better t_{CO} performance.

Intel devices have a variety of hierarchical clock structures. These include dedicated global clock networks, regional clock networks, fast regional clock networks, and periphery clock networks. The available resources differ between the various Intel device families.

For the number of clocking resources available in your target device, refer to the appropriate device handbook.

9.6.3.7 Spine Clock Limitations

If your project has high clock routing demands, due to limitations in the Quartus Prime software, you may see spine clock errors. These errors are often seen with designs using multiple memory interfaces and high-speed serial interface (HSSI) channels (especially PMA Direct mode).

Global clock networks, regional clock networks, and periphery clock networks have an additional level of clock hierarchy known as spine clocks. Spine clocks drive the final row and column clocks to their registers; thus, the clock to every register in the chip is reached through spine clocks. Spine clocks are not directly user controllable.

To reduce these spine clock errors, constrain your design to better use your regional clock resources:

- If your design does not use LogicLock Plus regions, or if the LogicLock Plus regions are not aligned to your clock region boundaries, create additional LogicLock Plus regions and further constrain your logic.

Note: Register packing, a Fitter optimization option, may ignore LogicLock Plus regions. If this occurs, disable register packing for specific instances through the Quartus Prime Assignment Editor.

- Some periphery features may ignore LogicLock Plus region assignment, possibly because the global promotion process doesn't function properly. To ensure that the global promotion process uses the correct locations, assign specific pins to the I/Os using these periphery features.
- By default, some IP MegaCore functions apply a global signal assignment with a value of dual-regional clock. If you constrain your logic to a regional clock region and set the global signal assignment to **Regional** instead of **Dual-Regional**, you can reduce clock resource contention.

Related Links

- Viewing Available Clock Networks in the Device on page 193
  When you enable a clock region layer in the **Layers Settings** pane, you display the areas of the chip that are driven by global and regional clock networks.
- Layers Settings on page 191
  The Chip Planner allows you to control the display of resources.
- Report Spine Clock Utilization dialog box (Chip Planner)
  In Quartus Prime Help

9.6.4 Register-to-Register Timing Optimization Techniques

The next stage of design optimization seeks to improve register-to-register (f_{MAX}) timing. The following sections provide available options if the performance requirements are not achieved after compilation.

Coding style affects the performance of your design to a greater extent than other changes in settings. Always evaluate your code and make sure to use synchronous design practices.

Note: When using the TimeQuest analyzer, register-to-register timing optimization is the same as maximizing the slack on the clock domains in your design. You can use the techniques described in this section to improve the slack on different timing paths in your design.

Before optimizing your design, understand the structure of your design as well as the type of logic affected by each optimization. An optimization can decrease performance if the optimization does not benefit your logic structure.

Related Links

Recommended Design Practices
In *Quartus Prime Pro Edition Handbook Volume 1*

9.6.4.1 Optimize Source Code

In many cases, optimizing the design's source code can have a very significant effect on your design performance. In fact, optimizing your source code is typically the most effective technique for improving the quality of your results and is often a better choice than using LogicLock Plus or location assignments.

Be aware of the number of logic levels needed to implement your logic while you are coding. Too many levels of logic between registers could result in critical paths failing timing. Try restructuring the design to use pipelining or more efficient coding techniques. Also, try limiting high fan-out signals in the source code. When possible, duplicate and pipeline control signals. Make sure the duplicate registers are protected by a preserve attribute, to avoid merging during synthesis.

If the critical path in your design involves memory or DSP functions, check whether you have code blocks in your design that describe memory or functions that are not being inferred and placed in dedicated logic. You might be able to modify your source code to cause these functions to be placed into high-performance dedicated memory or resources in the target device. When using RAM/DSP blocks, enable the optional input and output registers.

Ensure that your state machines are recognized as state machine logic and optimized appropriately in your synthesis tool. State machines that are recognized are generally optimized better than if the synthesis tool treats them as generic logic. In the Quartus Prime software, you can check the State Machine report under **Analysis & Synthesis** in the Compilation Report. This report provides details, including state encoding for each state machine that was recognized during compilation. If your state machine is not recognized, you might have to change your source code to enable it to be recognized.

Related Links

- Recommended HDL Coding Styles
  In *Quartus Prime Pro Edition Handbook Volume 1*
- AN 584: Timing Closure Methodology for Advanced FPGA Designs

9.6.4.2 Improving Register-to-Register Timing

The choice of options and settings to improve the timing margin (slack) or to improve register-to-register timing depends on the failing paths in the design. To achieve the results that best approximate your performance requirements, apply the following techniques and compile the design after each step:

1. Ensure that your timing assignments are complete and correct. For details, refer to the *Initial Compilation: Required Settings* section in the *Design Optimization Overview* chapter.
2. Ensure that you have reviewed all warning messages from your initial compilation and check for ignored timing assignments.
3. Apply netlist synthesis optimization options.
4. To optimize for speed, apply the following synthesis options:

- Optimize Synthesis for Speed, Not Area
- Flatten the Hierarchy During Synthesis
- Set the Synthesis Effort to High
- Prevent Shift Register Inference
- Use Other Synthesis Options Available in Your Synthesis Tool

5. To optimize for performance using physical synthesis, apply the following options:
   - Enable physical synthesis
   - Perform automatic asynchronous signal pipelining
   - Perform register duplication
   - Perform register retiming
   - Perform logic to memory mapping
6. Try different Fitter seeds. If there are very few paths that are failing by small negative slack, then you can try with a different seed to see if there is a fit that meets constraints in the Fitter seed noise.

   Note: Omit this step if a large number of critical paths are failing or if the paths are failing badly.
7. To control placement, make LogicLock Plus assignments.
8. Make design source code modifications to fix areas of the design that are still failing timing requirements by significant amounts.
9. Make location assignments, or as a last resort, perform manual placement by back-annotating the design.

   You can use Design Space Explorer II (DSE) to automate the process of running several different compilations with different settings.

   If these techniques do not achieve performance requirements, additional design source code modifications might be required.

Related Links

- Launch Design Space Explorer Command (Tools Menu)
  In Quartus Prime Help
- Initial Compilation: Required Settings
  In *Quartus Prime Pro Edition Handbook Volume 1*

9.6.4.3 Physical Synthesis Optimizations

The Quartus Prime software offers physical synthesis optimizations that can help improve the performance of many designs, regardless of the synthesis tool used. Physical synthesis optimizations can be applied both during synthesis and during fitting.

Physical synthesis optimizations that occur during the synthesis stage of the Quartus Prime compilation operate either on the output from another EDA synthesis tool or as an intermediate step in Quartus Prime synthesis. These optimizations make changes to the synthesis netlist to improve either area or speed, depending on your selected optimization technique and effort level.

To view and modify the synthesis netlist optimization options, click **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Fitter)**.

If you use a third-party EDA synthesis tool and want to determine if the Quartus Prime software can remap the circuit to improve performance, you can use the **Perform WYSIWYG Primitive Resynthesis** option. This option directs the Quartus Prime software to unmap the LEs in an atom netlist to logic gates and then map the gates back to Intel-specific primitives. Using Intel-specific primitives enables the Fitter to remap the circuits using architecture-specific techniques.

The Quartus Prime technology mapper optimizes the design to achieve maximum speed performance, minimum area usage, or balances high performance and minimal logic usage, according to the setting of the **Optimization Technique** option. Set this option to **Speed** or **Balanced**.

The physical synthesis optimizations occur during the Fitter stage of the Quartus Prime compilation. Physical synthesis optimizations make placement-specific changes to the netlist that improve speed performance results for a specific Intel device.

Note: If you want the performance gain from physical synthesis only on parts of your design, you can apply the physical synthesis options on specific instances.

To apply physical synthesis assignments for fitting on a per-instance basis, use the Quartus Prime Assignment Editor. The following assignments are available as instance assignments:

Related Links

- Perform WYSIWYG Primitive Resynthesis Logic Option
  In Quartus Prime Help
- Optimization Technique Logic Option
  In Quartus Prime Help

9.6.4.4 Turn Off Extra-Effort Power Optimization Settings

If power optimization settings are set to **Extra Effort**, your design performance can be affected. If timing performance is more important than power, set the power optimization setting to **Normal**.

Related Links

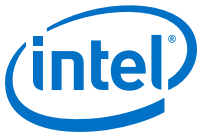
- Power Optimization Logic Option
  In Quartus Prime Help
- Power Optimization on page 158

9.6.4.5 Optimize Synthesis for Speed, Not Area

Design performance varies depending on coding style, synthesis tool used, and options specified when synthesizing. Change your synthesis options if a large number of paths are failing or if specific paths are failing badly and have many levels of logic.

Set your device and timing constraints in your synthesis tool. Synthesis tools are timing-driven and optimized to meet specified timing requirements. If you do not specify a target frequency, some synthesis tools optimize for area.

Some synthesis tools offer an easy way to instruct the tool to focus on speed instead of area.



You can also specify this logic option for specific modules in your design with the Assignment Editor while leaving the default **Optimization Technique** setting at **Balanced** (for the best trade-off between area and speed for certain device families) or **Area** (if area is an important concern). You can also use the **Speed Optimization Technique for Clock Domains** option in the Assignment Editor to specify that all combinational logic in or between the specified clock domain(s) is optimized for speed.

To achieve best performance with push-button compilation, follow the recommendations in the following sections for other synthesis settings. You can use DSE II to experiment with different Quartus Prime synthesis options to optimize your design for the best performance.

Related Links

Optimization Technique Logic Option
In Quartus Prime Help

9.6.4.6 Flatten the Hierarchy During Synthesis

Synthesis tools typically let you preserve hierarchical boundaries, which can be useful for verification or other purposes. However, the best optimization results generally occur when the synthesis tool optimizes across hierarchical boundaries, because doing so often allows the synthesis tool to perform the most logic minimization, which can improve performance. Whenever possible, flatten your design hierarchy to achieve the best results.

9.6.4.7 Set the Synthesis Effort to High

Some synthesis tools offer varying synthesis effort levels to trade off compilation time with synthesis results. Set the synthesis effort to **high** to achieve best results when applicable.

9.6.4.8 Duplicate Logic for Fan-Out Control

Oftentimes, timing failures occur not because of the high fan-out registers, but because of the location of those registers. Duplicating registers, where source and destination registers are physically close, can help improve slack on critical paths.

Many synthesis tools support options or attributes that specify the maximum fan-out of a register. When using Quartus Prime synthesis, you can set the **Maximum Fan-Out** logic option in the Assignment Editor to control the number of destinations for a node so that the fan-out count does not exceed a specified value. You can also use the `maxfan` attribute in your HDL code. The software duplicates the node as required to achieve the specified maximum fan-out.

Logic duplication using **Maximum Fan-Out** assignments normally increases resource utilization and can potentially increase compilation time, depending on the placement and the total resource usage within the selected device. The improvement in timing performance that results because of **Maximum Fan-Out** assignments is very design-specific. This is because when you use the **Maximum Fan-Out** assignment, although the Fitter duplicates the source logic to limit the fan-out, it may not be able to control the destinations that each of the duplicated sources drive. Since the **Maximum Fan-Out** destination does not specify which of the destinations the duplicated source should drive, it is possible that it might still be driving logic located all around the device. To avoid this situation, you could use the **Manual Logic Duplication** logic option.

If you are using **Maximum Fan-Out** assignments, Intel recommends benchmarking your design with and without these assignments to evaluate whether they give the expected improvement in timing performance. Use the assignments only when you get improved results.

You can manually duplicate registers in the Quartus Prime software regardless of the synthesis tool used. To duplicate a register, apply the **Manual Logic Duplication** logic option to the register with the Assignment Editor.

Note: Various Fitter optimizations may cause a small violation to the **Maximum Fan-Out** assignments to improve timing.

Related Links

Manual Logic Duplication Logic Option
In Quartus Prime Help

9.6.4.9 Prevent Shift Register Inference

In some cases, turning off the inference of shift registers increases performance. Doing so forces the software to use logic cells to implement the shift register instead of implementing the registers in memory blocks using the ALTSHIFT_TAPS IP core. If you implement shift registers in logic cells instead of memory, logic utilization is increased.

9.6.4.10 Use Other Synthesis Options Available in Your Synthesis Tool

With your synthesis tool, experiment with the following options if they are available:

- Turn on register balancing or retiming
- Turn on register pipelining
- Turn off resource sharing

These options can increase performance, but typically increase the resource utilization of your design.

9.6.4.11 Fitter Seed

The Fitter seed affects the initial placement configuration of the design. Changing the seed value changes the Fitter results because the fitting results change whenever there is a change in the initial conditions. Each seed value results in a somewhat different fit, and you can experiment with several different seeds to attempt to obtain better fitting results and timing performance.

When there are changes in your design, there is some random variation in performance between compilations. This variation is inherent in placement and routing algorithms—there are too many possibilities to try them all and get the absolute best result, so the initial conditions change the compilation result.

Note: Any design change that directly or indirectly affects the Fitter has the same type of random effect as changing the seed value. This includes any change in source files, **Compiler Settings** or **Timing Analyzer Settings**. The same effect can appear if you use a different computer processor type or different operating system, because different systems can change the way floating point numbers are calculated in the Fitter.

If a change in optimization settings slightly affects the register-to-register timing or number of failing paths, you cannot always be certain that your change caused the improvement or degradation, or whether it could be due to random effects in the Fitter. If your design is still changing, running a seed sweep (compiling your design with multiple seeds) determines whether the average result has improved after an optimization change and whether a setting that increases compilation time has benefits worth the increased time, such as with physical synthesis settings. The sweep also shows the amount of random variation to expect for your design.

If your design is finalized, you can compile your design with different seeds to obtain one optimal result. However, if you subsequently make any changes to your design, you might need to perform seed sweep again.

Click **Assignments ➤ Compiler Settings** to control the initial placement with the seed. You can use the DSE II to perform a seed sweep easily.

You can use the following Tcl command from a script to specify a Fitter seed:

```
set_global_assignment -name SEED <value>
```

Related Links

Launch Design Space Explorer Command (Tools Menu)
In Quartus Prime Help

9.6.4.12 Set Maximum Router Timing Optimization Level

To improve routability in designs where the router did not pick up the optimal routing lines, set the **Router Timing Optimization Level** to **Maximum**. This setting determines how aggressively the router tries to meet the timing requirements. Setting this option to **Maximum** can increase design speed slightly at the cost of increased compilation time. Setting this option to **Minimum** can reduce compilation time at the cost of slightly reduced design speed. The default value is **Normal**.

Related Links

Router Timing Optimization Level Logic Option
In Quartus Prime Help

9.6.5 Location Assignments

If a small number of paths are failing to meet their timing requirements, you can use hard location assignments to optimize placement. Location assignments are less flexible for the Quartus Prime Fitter than LogicLock Plus assignments. In some cases, when you are familiar with your design, you can enter location constraints in a way that produces better results.

Note: Improving fitting results, especially for larger devices, such as Arria and Stratix series devices, can be difficult. Location assignments do not always improve the performance of the design. In many cases, you cannot improve upon the results from the Fitter by making location assignments.

9.6.6 Metastability Analysis and Optimization Techniques

Metastability problems can occur when a signal is transferred between circuitry in unrelated or asynchronous clock domains, because the designer cannot guarantee that the signal will meet its setup and hold time requirements.The mean time between failures (MTBF) is an estimate of the average time between instances when metastability could cause a design failure.

You can use the Quartus Prime software to analyze the average MTBF due to metastability when a design synchronizes asynchronous signals and to optimize the design to improve the MTBF. These metastability features are supported only for designs constrained with the TimeQuest analyzer, and for select device families.

If the MTBF of your design is low, refer to the Metastability Optimization section in the Timing Optimization Advisor, which suggests various settings that can help optimize your design in terms of metastability.

This chapter describes how to enable metastability analysis and identify the register synchronization chains in your design, provides details about metastability reports, and provides additional guidelines for managing metastability.

Related Links

- Understanding Metastability in FPGAs
- Managing Metastability with the Quartus Prime Software
  In *Quartus Prime Pro Edition Handbook Volume 1*

9.7 Periphery to Core Register Placement and Routing Optimization

The Periphery to Core Register Placement and Routing Optimization (P2C) option specifies whether the Fitter performs targeted placement and routing optimization on direct connections between periphery logic and registers in the FPGA core.P2C is an optional pre-routing-aware placement optimization stage that enables you to more reliably achieve timing closure.

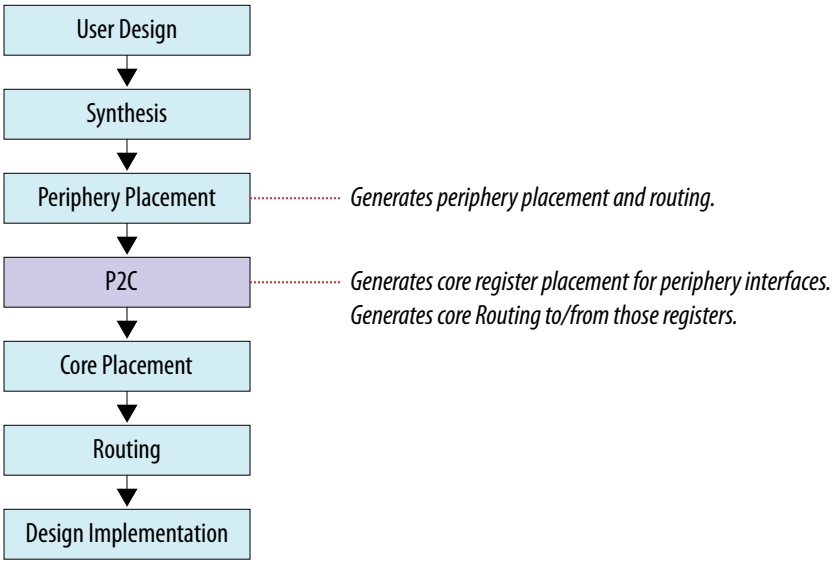
Note: The **Periphery to Core Register Placement and Routing Optimization** option applies in both directions, periphery to core and core to periphery.

Transfers between external interfaces (for example, high-speed I/O or serial interfaces) and the FPGA often require routing many connections with tight setup and hold timing requirements. When this option is turned on, the Fitter performs P2C placement and routing decisions before those for core placement and routing. This reserves the necessary resources to ensure that your design achieves its timing requirements and avoids routing congestion for transfers with external interfaces.

This option is available as a global assignment, or can be applied to specific instances within your design.

Figure 42. Periphery to Core Register Placement and Routing Optimization (P2C) Flow

P2C runs after periphery placement, and generates placement for core registers on corresponding P2C/C2P paths, and core routing to and from these core registers.



Setting Periphery to Core Optimizations in the Advanced Fitter Setting Dialog Box on page 139
The **Periphery to Core Placement and Routing Optimization** setting specifies whether the Fitter should perform targeted placement and routing optimization on direct connections between periphery logic and registers in the FPGA core.

Setting Periphery to Core Optimizations in the Assignment Editor on page 140
When you turn on the **Periphery to Core Placement and Routing Optimization** (P2C/C2P) setting in the Assignment Editor, the Quartus Prime software performs periphery to core, or core to periphery optimizations on selected instances in your design.

Viewing Periphery to Core Optimizations in the Fitter Report on page 140
The Quartus Prime software generates a periphery to core placement and routing optimization summary in the **Fitter (Place & Route)** report after compilation.

9.7.1 Setting Periphery to Core Optimizations in the Advanced Fitter Setting Dialog Box

The **Periphery to Core Placement and Routing Optimization** setting specifies whether the Fitter should perform targeted placement and routing optimization on direct connections between periphery logic and registers in the FPGA core.

You can optionally perform periphery to core optimizations by instance with settings in the Assignment Editor.

1. In the Quartus Prime software, click **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Fitter)**.
2. In the **Advanced Fitter Settings** dialog box, for the **Periphery to Core Placement and Routing Optimization** option, select one of the following options depending on how you want to direct periphery to core optimizations in your design:

a. Select **Auto** to direct the software to automatically identify transfers with tight timing windows, place the core registers, and route all connections to or from the periphery.

b. Select **On** to direct the software to globally optimize all transfers between the periphery and core registers, regardless of timing requirements.

Note: Setting this option to **On** in the **Advanced Fitter Settings** is not recommended. The intended use for this setting is in the Assignment Editor to force optimization for a targeted set of nodes or instance.

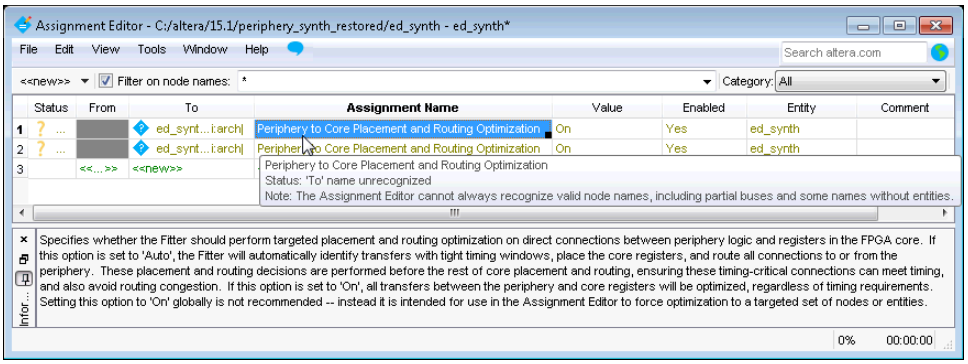
c. Select **Off** to disable periphery to core path optimization in your design.

9.7.2 Setting Periphery to Core Optimizations in the Assignment Editor

When you turn on the **Periphery to Core Placement and Routing Optimization** (P2C/C2P) setting in the Assignment Editor, the Quartus Prime software performs periphery to core, or core to periphery optimizations on selected instances in your design.

You can optionally perform periphery to core optimizations by instance with settings in the **Advanced Fitter Settings** dialog box.

1. In the Quartus Prime software, click **Assignments ➤ Assignment Editor**.
2. For the selected path, double-click the **Assignment Name** column, and then click the **Periphery to core register placement and routing optimization** option in the drop-down list.
3. In the **To** column, choose either a periphery node or core register node on a P2C/C2P path you want to optimize. Leave the **From** column empty.
   For paths to appear in the Assignments Editor, you must first run Analysis & Synthesis on your design.



9.7.3 Viewing Periphery to Core Optimizations in the Fitter Report

The Quartus Prime software generates a periphery to core placement and routing optimization summary in the **Fitter (Place & Route)** report after compilation.

1. Compile your Quartus Prime project.
2. In the **Tasks** pane, select **Compilation**.
3. Under **Fitter (Place & Route)**, double-click **View Report**.
4. In the **Fitter** folder, expand the **Place Stage** folder.
5. Double-click **Periphery to Core Transfer Optimization Summary**.

Table 37. Fitter Report - Periphery to Core Transfer Optimization (P2C) Summary

From Path	To Path	Status
Node 1	Node 2	**Placed and Routed**—Core register is locked. Periphery to core/core to periphery routing is committed.
Node 3	Node 4	**Placed but not Routed**—Core register is locked. Routing is not committed. This occurs when P2C is not able to optimize all targeted paths within a single group, for example, the same delay/wire requirement, or the same control signals. Partial P2C routing commitments may cause unresolvable routing congestion.
Node 5	Node 6	**Not Optimized**—This occurs when P2C is set to **Auto** and the path is not optimized due to one of the following issues: a. The delay requirement is impossible to achieve. b. The minimum delay requirement (for hold timing) is too large. The P2C algorithm cannot efficiently handle cases when many wires need to be added to meet hold timing. c. P2C encountered unresolvable routing congestion for this particular path.

Periphery to Core Transfer Optimization Summary

	From	To	Status
1	Periphery to Core Transfer		
2	dut|arch|arch_inst|io_tiles_wrap_inst|io_tiles_inst|tile_gen[1].lane_gen[3].lane_inst	dut|arch|arch_inst|afi.if_inst|single_port.afi_rdata_valid_regs|sr_out[0]	Placed but Not Routed
3	dut|arch|arch_inst|io_tiles_wrap_inst|io_tiles_inst|tile_gen[2].lane_gen[2].lane_inst	dut|arch|arch_inst|afi.if_inst|mem_sp_bidir_data.mem_dq.afi_regs_i|sr_out[63]	Placed but Not Routed
4	dut|arch|arch_inst|io_tiles_wrap_inst|io_tiles_inst|tile_gen[0].lane_gen[1].lane_inst	dut|arch|arch_inst|afi.if_inst|mem_sp_bidir_data.mem_dq.afi_regs_i|sr_out[11]	Placed but Not Routed
5	dut|arch|arch_inst|io_tiles_wrap_inst|io_tiles_inst|tile_gen[0].lane_gen[0].lane_inst	dut|arch|arch_inst|afi.if_inst|mem_sp_bidir_data.mem_dq.afi_regs_i|sr_out[7]	Placed but Not Routed
6	dut|arch|arch_inst|io_tiles_wrap_inst|io_tiles_inst|tile_gen[0].lane_gen[0].lane_inst	dut|arch|arch_inst|afi.if_inst|mem_sp_bidir_data.mem_dq.afi_regs_i|sr_out[3]	Placed but Not Routed
7	dut|arch|arch_inst|io_tiles_wrap_inst|io_tiles_inst|tile_gen[0].lane_gen[0].lane_inst	dut|arch|arch_inst|afi.if_inst|mem_sp_bidir_data.mem_dq.afi_regs_i|sr_out[0]	Placed but Not Routed
8	dut|arch|arch_inst|io_tiles_wrap_inst|io_tiles_inst|tile_gen[0].lane_gen[0].lane_inst	dut|arch|arch_inst|afi.if_inst|mem_sp_bidir_data.mem_dq.afi_regs_i|sr_out[79]	Placed but Not Routed
9	dut|arch|arch_inst|io_tiles_wrap_inst|io_tiles_inst|tile_gen[0].lane_gen[0].lane_inst	dut|arch|arch_inst|afi.if_inst|mem_sp_bidir_data.mem_dq.afi_regs_i|sr_out[75]	Placed but Not Routed
10	dut|arch|arch_inst|io_tiles_wrap_inst|io_tiles_inst|tile_gen[0].lane_gen[0].lane_inst	dut|arch|arch_inst|afi.if_inst|mem_sp_bidir_data.mem_dq.afi_regs_i|sr_out[73]	Placed but Not Routed
11	dut|arch|arch_inst|io_tiles_wrap_inst|io_tiles_inst|tile_gen[2].lane_gen[3].lane_inst	dut|arch|arch_inst|afi.if_inst|mem_sp_bidir_data.mem_dq.afi_regs_i|sr_out[71]	Placed but Not Routed
12	dut|arch|arch_inst|io_tiles_wrap_inst|io_tiles_inst|tile_gen[2].lane_gen[3].lane_inst	dut|arch|arch_inst|afi.if_inst|mem_sp_bidir_data.mem_dq.afi_regs_i|sr_out[69]	Placed but Not Routed
13	dut|arch|arch_inst|io_tiles_wrap_inst|io_tiles_inst|tile_gen[1].lane_gen[3].lane_inst	dut|arch|arch_inst|afi.if_inst|mem_sp_bidir_data.mem_dq.afi_regs_i|sr_out[35]	Placed but Not Routed
14	dut|arch|arch_inst|io_tiles_wrap_inst|io_tiles_inst|tile_gen[0].lane_gen[3].lane_inst	dut|arch|arch_inst|afi.if_inst|mem_sp_bidir_data.mem_dq.afi_regs_i|sr_out[31]	Placed but Not Routed
15	dut|arch|arch_inst|io_tiles_wrap_inst|io_tiles_inst|tile_gen[0].lane_gen[3].lane_inst	dut|arch|arch_inst|afi.if_inst|mem_sp_bidir_data.mem_dq.afi_regs_i|sr_out[27]	Placed but Not Routed
16	dut|arch|arch_inst|io_tiles_wrap_inst|io_tiles_inst|tile_gen[0].lane_gen[2].lane_inst	dut|arch|arch_inst|afi.if_inst|mem_sp_bidir_data.mem_dq.afi_regs_i|sr_out[23]	Placed but Not Routed

9.8 Design Evaluation for Timing Closure

Follow the guidelines in this section when you encounter timing failures in a design. The guidelines show you how to evaluate compilation results of a design and how to address some of the problems. While the guideline does not cover specific examples of restructuring RTL to improve design speed, the analysis techniques help you to evaluate changes that may have to be made to RTL to close timing.

9.8.1 Review Compilation Results

9.8.1.1 Review Messages

After compiling your design, review the messages in each section of the compilation report. Most designs that fail timing start out with other problems that are reported as warning messages during compilation. Determine what causes a warning message, and whether the warning should be fixed or ignored. After reviewing the warning

messages, review the informational messages. Take note of anything unexpected, for example, unconnected ports, ignored constraints, missing files, and assumptions or optimizations that the software made.

9.8.1.2 Evaluate Fitter Netlist Optimizations

The Fitter can also perform netlist optimizations to the design netlist. Major changes include register packing, duplicating or deleting logic cells, retiming registers, inverting signals, or modifying nodes in a general way such as moving an input from one logic cell to another. These reports can be found in the Netlist Optimizations results of the Fitter section, and they should also be reviewed.

9.8.1.3 Evaluate Optimization Results

After checking what optimizations were done and how they improved performance, evaluate the runtime it took to get the extra performance. To reduce compilation time, review the physical synthesis and netlist optimizations over a couple of compilations, and edit the RTL to reflect the changes that physical synthesis performed. If a particular set of registers consistently get retimed, edit the RTL to retime the registers the same way. If the changes are made to match what the physical synthesis algorithms did, the physical synthesis options can be turned off to save compile time while getting the same type of performance improvement.

9.8.1.4 Evaluate Resource Usage

Evaluate a variety of resources used in the design, including global and non-global signal usage, routing utilization, and clustering difficulty.

9.8.1.4.1 Global and Non-global Usage

If your design contains a lot of clocks, evaluate global and non-global signals. Determine whether global resources are being used effectively, and if not, consider making changes. These reports can be found in the Resource Section under Fitter in the Compilation Report panel. The figure shows an example of inefficient use of a global clock. The highlighted line has a single fan-out from a global clock. Assigning it to a Regional Clock would make the Global Clock available for another signal. You can ignore signals with an empty value in the **Global Line Name** column as the signal uses dedicated routing, and not a clock buffer.

Figure 43. Inefficient Use of a Global Clock

Global & Other Fast Signals

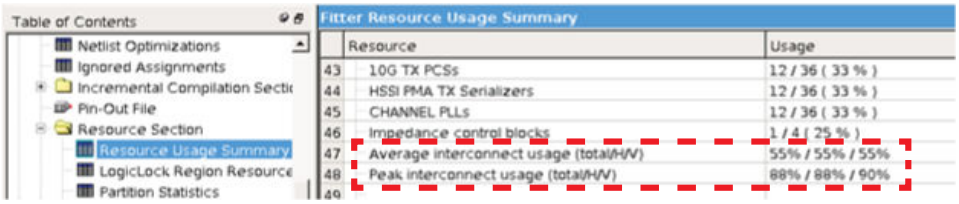
Location	Fan-Out	Global Resource Used	Global Line Name
FRACTIONALPLL_X98_Y2_N0	1	Global Clock	--
PLLOUTPUTCOUNTER_X98_Y2_N1	29044	Global Clock	GCLK7
PLLOUTPUTCOUNTER_X98_Y13_N1	253103	Global Clock	GCLK6
FF_X185_Y66_N13	280349	Global Clock	GCLK8
PIN_AE17	4887	Global Clock	GCLK4
FRACTIONALPLL_X98_Y11_N0	1	Global Clock	--
PLLOUTPUTCOUNTER_X98_Y3_N1	1	Global Clock	GCLK5
PLLOUTPUTCOUNTER_X98_Y1_N1	1691	Regional Clock	RCLK29
PLLOUTPUTCOUNTER_X98_Y8_N1	302	Regional Clock	RCLK23
PLLOUTPUTCOUNTER_X98_Y11_N1	141	Regional Clock	RCLK25
PLLOUTPUTCOUNTER_X98_Y10_N1	17	Regional Clock	RCLK22

The Non-Global High Fan-Out Signals report lists the highest fan-out nodes that are not routed on global signals. Reset and enable signals are at the top of the list. If there is routing congestion in the design, and there are high fan-out non-global nodes in the congested area, consider using global or regional signals to fan-out the nodes, or duplicate the high fan-out registers so that each of the duplicates can have fewer fan-outs. Use the Chip Planner to locate high fan-out nodes, to report routing congestion, and to determine whether the alternatives are viable.

9.8.1.4.2 Routing Usage

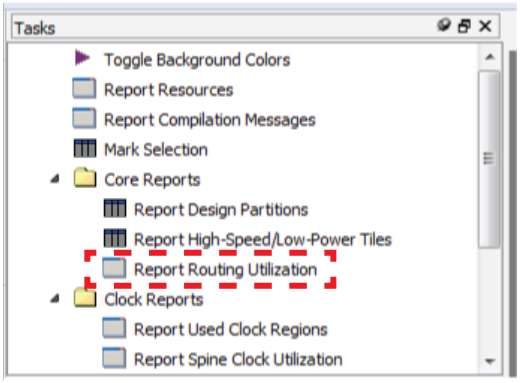
Review routing usage reported in the **Fitter Resource Usage Summary** report. The figure shows an example of the report.

Figure 44. Fitter Resource Usage Summary Report



The average interconnect usage reports the average amount of interconnect that is used, out of what is available on the device. The peak interconnect usage reports the largest amount of interconnect used in the most congested areas. Designs with an average value below 50% typically do not have any problems with routing. Designs with an average between 50-65% may have difficulty routing. Designs with an average over 65% typically have difficulty meeting timing unless the RTL is well designed to tolerate a highly utilized chip. Peak values at or above 90% are likely to have problems with timing closure; a 100% peak value indicates that all routing in an area of the device has been used, so there is a high possibility of degradation in timing performance. The figure shows the **Report Routing Utilization** report.

Figure 45. Report Routing Utilization Report

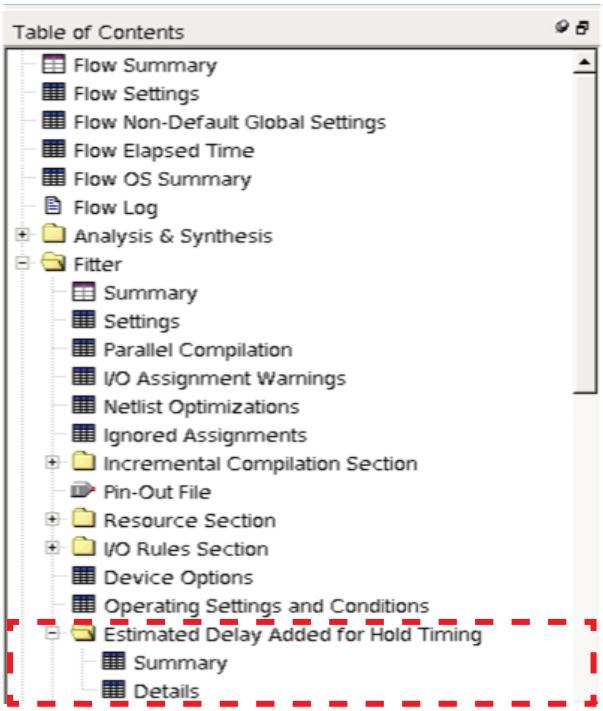


9.8.1.4.3 Wires Added for Hold

As part of the fitting process, the router can add wire between register paths to increase delay to meet hold time requirements. During the routing process, the router reports how much extra wire was used to meet hold time requirements. Excessive amounts of added wire can indicate problems with the constraint. Typically it would be caused by incorrect multicycle transfers, particularly between different rate clocks, and

between different clock networks. The Fitter reports how much routing delay was added in the **Estimated Delay Added for Hold Timing** report. Specific register paths can be reviewed to view whether a delay was added to meet hold requirements.

Figure 46. Estimated Delay Added for Hold Timing Report

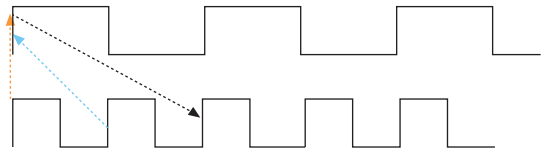


An example of an incorrect constraint which can cause the router to add wire for hold requirements is when there is data transfer from 1x to 2x clocks. Assume the design intent is to allow two cycles per transfer. Data can arrive any time in the two destination clock cycles by adding a multicycle setup constraint as shown in the example:

```
set_multicycle_path -from 1x -to 2x -setup -end 2
```

The timing requirement is relaxed by one 2x clock cycle, as shown in the black line in the waveform in the figure.

Figure 47. Timing Requirement Relaxed Waveform



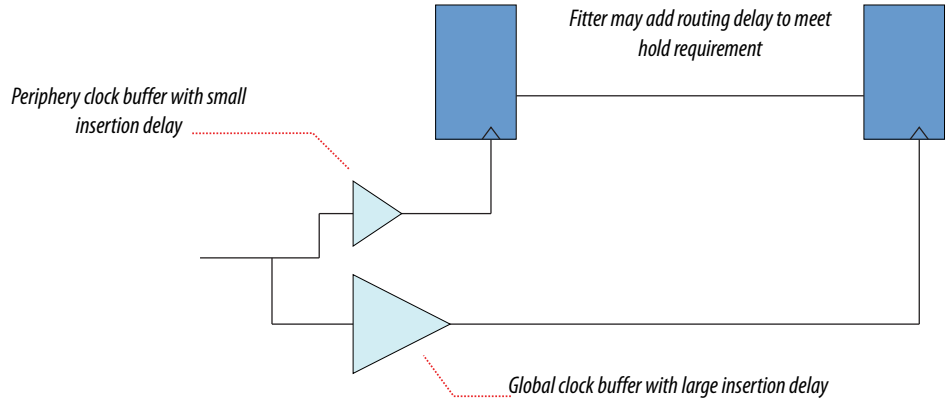
However, the default hold requirement, shown with the dashed blue line, may cause the router to add wire to guarantee that data is delayed by one cycle. To correct the hold requirement, add a multicycle constraint with a hold option.

```
set_multicycle_path -from 1x -to 2x -setup -end 2
set_multicycle_path -from 1x -to 2x -hold -end 1
```

The orange dashed line in the figure above represents the hold relationship, and no extra wire is required to delay the data.

The router can also add wire for hold timing requirements when data is transferred in the same clock domain, but between clock branches that use different buffering. Transferring between clock network types happens more often between the periphery and the core. The figure below shows a case where data is coming into a device, and uses a periphery clock to drive the source register, and a global clock to drive the destination register. A global clock buffer has larger insertion delay than a periphery clock buffer. The clock delay to the destination register is much larger than to the source register, hence extra delay is necessary on the data path to ensure that it meets its hold requirement.

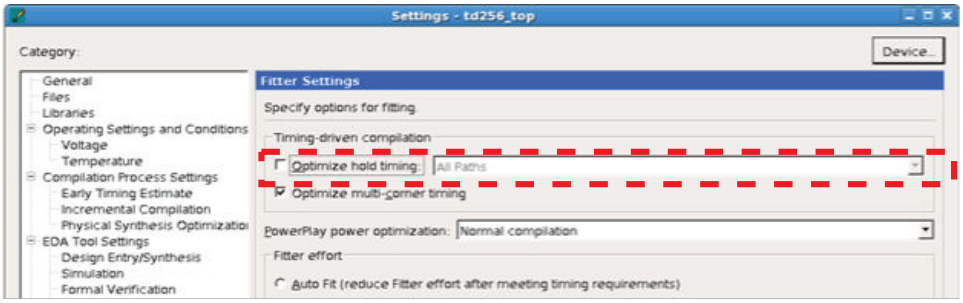
Figure 48. Clock Delay



To identify cases where a path has different clock network types, review the path in the TimeQuest timing analyzer, and check nodes along the source and destination clock paths. Also, check the source and destination clock frequencies to see whether they are the same, or multiples, and whether there are multicycle exceptions on the paths. In some cases, cross-domain paths may also be false by intent, so make sure there are false path exceptions on those.

If you suspect that routing is added to fix real hold problems, then disable the **Optimize hold timing** option. Recompile the design and rerun timing analysis to uncover paths that fail hold time.

Figure 49. Optimize Hold Timing Option



Disabling the **Optimize hold timing** option is a debug step, and should be left enabled (default state) during normal compiles. Wire added for hold is a normal part of timing optimization during routing and is not always a problem.

9.8.1.5 Evaluate Other Reports and Adjust Settings Accordingly

9.8.1.5.1 Difficulty Packing Design

In the Fitter Resource Section, under the **Resource Usage Summary**, review the **Difficulty Packing Design** report. The **Difficulty Packing Design** report details the effort level (low, medium, or high) of the Fitter to fit the design into the device, partition, and LogicLock Plus region.

As the effort level of **Difficulty Packing Design** increases, timing closure gets harder. Going from medium to high can result in significant drop in performance or increase in compile time. Consider reducing logic to reduce packing difficulty.

9.8.1.5.2 Review Ignored Assignments

The **Compilation Report** includes details of any assignments ignored by the Fitter. Assignments typically get ignored if design names change, but assignments are not updated. Make sure any intended assignments are not being ignored.

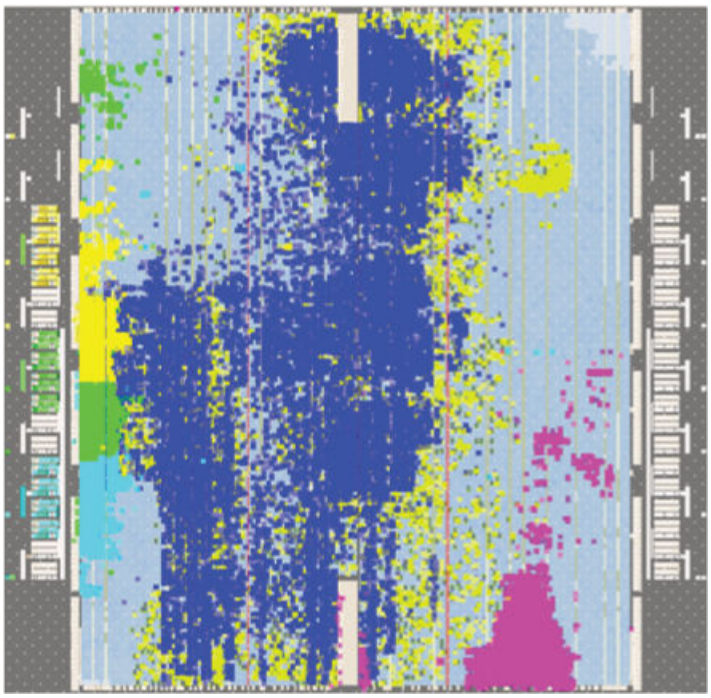
9.8.1.5.3 Review Non-Default Settings

The reports from Synthesis and Fitter show non-default settings used in a compilation. Review the non-default settings to ensure the design benefits from the change.

9.8.1.5.4 Review Floorplan

Use the Chip Planner for reviewing placement. The Chip Planner can be used to locate hierarchical entities, and colors each located entity in the floorplan. Look for logic that seems out of place, based on where you would expect it to be. For example, logic that interfaces with I/Os should be close to the I/Os, and logic that interfaces with an IP or memory must be close to the IP or memory. The figure shows an example of a floorplan with color-coded entities. In the floorplan, the green block is spread apart. Check to see if those paths are failing timing, and if so, what connects to that module that could affect placement. The blue and aqua blocks are spread out and mixed together. Check and see if there are many connections between the two modules that may contribute to this. The pink logic at the bottom should interface with I/Os at the bottom edge.

Figure 50. Floorplan with Color-Coded Entities



Check fan-in and fan-out of a highlighted module by using the buttons on the task bar shown in the figure below.

Figure 51. Fan-in and Fan-Out Buttons

Look for signals that go a long way across the chip and see if they are contributing to timing failures.

Check global signal usage for signals that may affect logic placement. Logic feeding a global buffer may be pulled close to the buffer, away from related logic. High fan-out on non-global resource may pull logic together.

Check for routing congestion. Highly congested areas may cause logic to be spread out, and the design may be difficult to route.

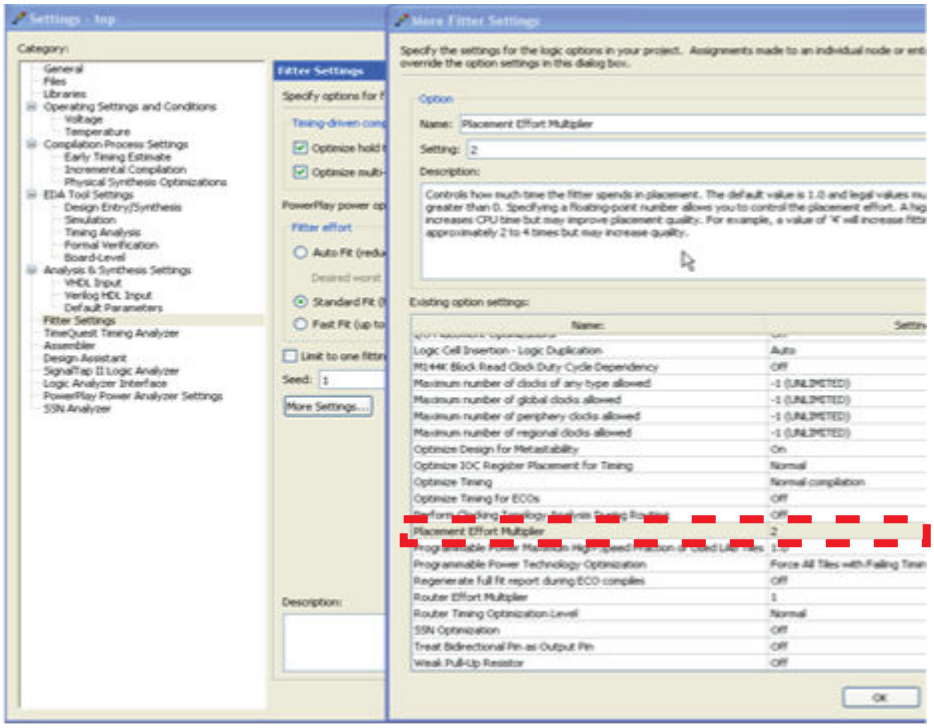
9.8.1.5.5 Evaluate Placement and Routing

Review duration of parts of compile time in Fitter messages. If routing takes much more time than placement, then meeting timing may be more difficult than the placer predicted.

9.8.1.5.6 Adjust Placement Effort

Increasing the **Placement Effort Multiplier** to improve placement quality may be a good tradeoff at the cost of higher compile time, but the benefit is design dependent. The value should be adjusted after reviewing and optimizing other settings and RTL. Try an increased value, up to 4, and reset to default if performance or compile time does not improve.

Figure 52. Placement Effort Multiplier



9.8.1.5.7 Review Timing Constraints

Ensure that clocks are constrained with the correct frequency requirements. Using the `derive_pll_clocks` assignment keeps generated clock settings updated. TimeQuest can be useful in reviewing SDC constraints. For example, under **Diagnostic** in the Task panel, the **Report Ignored Constraints** report shows any incorrect names in the design, most commonly caused by changes in the design hierarchy. Use the **Report Unconstrained Paths** report to locate unconstrained paths. Add constraints as necessary so that the design can be optimized.

9.8.1.6 Evaluate Clustering Difficulty

You can evaluate clustering difficulty to help reach timing closure.

You can monitor clustering difficulty whenever you add logic and recompile. Use the clustering information to gauge how much timing closure difficulty is inherent in your design:

- If your design is full but clustering difficulty is low or medium, your design itself, rather than clustering, is likely the main cause of congestion.
- Conversely, if congestion occurs after adding a small amount of logic to the design, this may be the result of clustering. If clustering difficulty is high, this contributes to congestion regardless of design size.

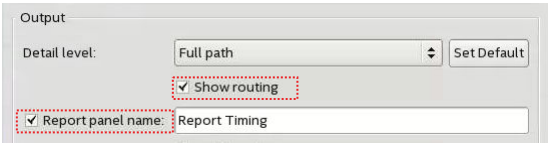
9.8.2 Review Details of Timing Paths

9.8.2.1 Show Timing Path Routing

Showing routing for a path can help uncover unusual routing delays.

In the TimeQuest **Report Timing** dialog box, enable the **Report panel name** and **Show routing** options, and click **Report Timing**.

Figure 53. Report Pane and Show Routing Options



The **Extra Fitter Information** tab shows a miniature floorplan with the path highlighted.

You can also locate the path in the Chip Planner to examine routing congestion, and to view whether nodes in a path are placed close together or far apart.

Related Links

Exploring Paths in the Chip Planner on page 196
Use the Chip Planner to explore paths between logic elements. The following examples use the Chip Planner to traverse paths from the Timing Analysis report.

9.8.2.2 Global Network Buffers

A routing path can be used to identify global network buffers that fail timing. Buffer locations are named according to the network they drive.

- CLK_CTRL_Gn—for Global driver
- CLk_CTRL_Rn—for Regional driver

Buffers to access the global networks are located in the center of each side of the device. The buffering to route a core logic signal on a global signal network will cause insertion delay. Some trade offs to consider for global and non-global routing are source location, insertion delay, fan-out, distance a signal travels, and possible congestion if the signal is demoted to local routing.

9.8.2.2.1 Source Location

If the register feeding the global buffer cannot be moved closer, then consider changing either the design logic or the routing type.

9.8.2.2.2 Insertion Delay

If a global signal is required, consider adding half a cycle to timing by using a negative-edge triggered register to generate the signal (top figure) and use a multicycle setup constraint (bottom figure).

Figure 54. Negative-Edge Triggered Register

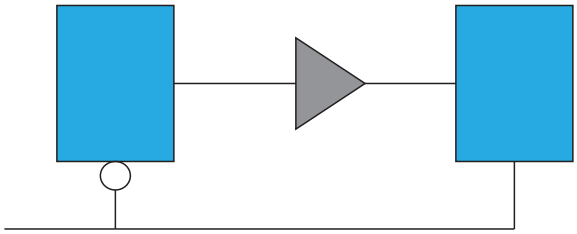
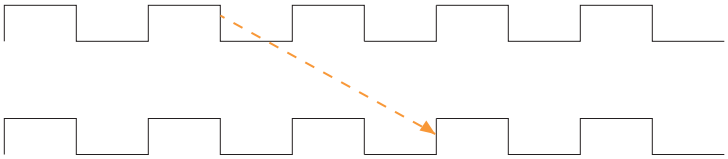


Figure 55. Multicycle Setup Constraint



```
set_multicycle_path -from <generating_register> -setup -end 2
```

9.8.2.2.3 Fan-Out

Nodes with very high fan-out that use local routing tend to pull logic that they drive close to the source node. This can make other paths fail timing. Duplicating registers can help reduce the impact of high fan-out paths. Consider manually duplicating and preserving these registers. Using a MAX_FANOUT assignment may make arbitrary groups of fan-out nodes, whereas a designer can make more intelligent fan-out groups.

9.8.2.2.4 Global Networks

If a signal should use a different type of global signal than it has automatically been assigned, use the Global Signal assignment to control the global signal usage on a per-signal basis. For example, if local routing is desired, set the Global Signal assignment to **OFF**.

Figure 56. Global Signal Assignment

To	Assignment Name	Value	Enabled
reg_clk	Global Signal	Off	Yes

9.8.2.3 Resets and Global Networks

Reset signals are often routed on global networks. Sometimes, the use of a global network causes recovery failures. Consider reviewing the placement of the register that generates the reset and the routing path of the signal.

9.8.2.4 Suspicious Setup

Suspicious setup failures include paths with very small or very large requirements. One typical cause is math precision error. For example, 10Mhz/3 = 33.33 ns per period. In three cycles, the time would be 99.999 ns vs 100.000 ns. Setting a maximum delay could provide an appropriate setup relationship.

Another cause of failure would be paths that should be false by design intent, such as:

- asynchronous paths that are handled through FIFOs, or
- slow asynchronous paths that rely on handshaking for data that remain available for multiple clock cycles.

To prevent the Fitter from having to meet unnecessarily restrictive timing requirements, consider adding false or multicycle path statements.

9.8.2.5 Logic Depth

The **Statistics** tab in the TimeQuest path report shows the levels of logic in a path. If the path fails timing and the number of logic levels is high, consider adding pipelining in that part of the design.

9.8.2.6 Auto Shift Register Replacement

Shift registers or register chains can be converted to RAM during synthesis to save area. However, conversion to RAM often reduces speed. The names of the converted registers will include "altshift_taps".

If paths that fail timing begin or end in shift registers, consider disabling the **Auto Shift Register Replacement** option. Registers that are intended for pipelining should not be converted. For shift registers that are converted to a chain, evaluate area/speed trade off of implementing in RAM or logic cells. If a design is close to full, shift register conversion to RAM may benefit non-critical clock domains by saving area. The settings can be changed globally or on a register or hierarchy basis from the default of **AUTO** to **OFF**.

9.8.2.7 Clocking Architecture

For better timing results, place registers driven by a regional clock in one quadrant of the chip. You can review the clock region boundaries using the Chip Planner.

Timing failure can occur when the I/O interface at the top of the device connects to logic driven by a regional clock which is in one quadrant of the device, and placement restrictions force long paths to and from some of the I/Os to logic across quadrants.

Use a different type of clock source to drive the logic - global, which covers the whole device, or dual-regional which covers half the device. Alternatively, you can reduce the frequency of the I/O interface to accommodate the long path delays. You can also redesign the pinout of the device to place all the specified I/Os adjacent to the regional clock quadrant. This issue can happen when register locations are restricted, such as with LogicLock Plus regions, clocking resources, or hard blocks (memories, DSPs, IPs).

The **Extra Fitter Information** tab in the TimeQuest timing report informs you when placement is restricted for nodes in a path.

Related Links

Viewing Available Clock Networks in the Device on page 193
When you enable a clock region layer in the **Layers Settings** pane, you display the areas of the chip that are driven by global and regional clock networks.

9.8.2.8 Timing Closure Recommendations

The Report Timing Closure Recommendations task in the TimeQuest analyzer analyzes paths and provides specific recommendations based on path characteristics.

9.8.3 Making Adjustments and Recompiling

Look for obvious problems that you can fix with minimal effort. To identify where the Compiler had trouble meeting timing, perform seed sweeping with about five compiles. Doing so shows consistently failing paths. Consider recoding or redesigning that part of the design.

To reach timing closure, a well written RTL can be more effective than changing your compilation settings. Seed sweeping can also be useful if the timing failure is very small, and the design has already been optimized for performance improvements and is close to final release. Additionally, seed sweeping can be used for evaluating changes to compilation settings. Compilation results vary due to the random nature of fitter algorithms. If a compilation setting change produces lower average performance, undo the change.

Sometimes, settings or constraints can cause more problems than they fix. When significant changes to the RTL or design architecture have been made, compile periodically with default settings and without LogicLock Plus regions, and re-evaluate paths that fail timing.

Partitioning often does not help timing closure, and should be done at the beginning of the design process. Adding partitions can increase logic utilization if it prevents cross-boundary optimizations, making timing closure harder and increasing compile times.

Adding LogicLock Plus regions can be an effective part of timing closure, but must be done at the beginning of a design. Adding new LogicLock Plus regions at the end of the design cycle can restrict placement, hence lowering the performance.

9.9 Scripting Support

You can run procedures and make settings described in this manual in a Tcl script. You can also run some procedures at a command prompt. For detailed information about scripting command options, refer to the Quartus Prime command-line and Tcl API Help browser. To run the Help browser, type the following command at the command prompt:

```
quartus_sh --qhelp
```

You can specify many of the options described in this section either in an instance, or at a global level, or both.

Use the following Tcl command to make a global assignment:

```
set_global_assignment -name <.qsf variable name> <value>
```

Use the following Tcl command to make an instance assignment:

```
set_instance_assignment -name <.qsf variable name> <value> -to <instance name>
```

Note: If the *<value>* field includes spaces (for example, 'Standard Fit'), you must enclose the value in straight double quotation marks.

Related Links

- Tcl Scripting on page 72
- Quartus Prime Settings Reference File Manual
- Command Line Scripting on page 65
  FPGA design software that easily integrates into your design flow saves time and improves productivity. The Intel Quartus Prime software provides you with a command-line executable for each step of the FPGA design flow to make the design process customizable and flexible.

9.9.1 Initial Compilation Settings

Use the Quartus Prime Settings File (`.qsf`) variable name in the Tcl assignment to make the setting along with the appropriate value. The **Type** column indicates whether the setting is supported as a global setting, an instance setting, or both.

The top table lists the `.qsf` variable name and applicable values for the settings described in the *Initial Compilation: Required Settings* section in the *Design Optimization Overview* chapter. The bottom table lists the advanced compilation settings.

Table 38. Initial Compilation Settings

Setting Name	.qsf File Variable Name	Values	Type
Optimize IOC Register Placement For Timing	`OPTIMIZE_IOC_REGISTER_PLACEMENT_FOR_TIMING`	`ON, OFF`	Global
Optimize Hold Timing	`OPTIMIZE_HOLD_TIMING`	`OFF, IO PATHS AND MINIMUM TPD PATHS, ALL PATHS`	Global

Table 39. Advanced Compilation Settings

Setting Name	.qsf File Variable Name	Values	Type
Router Timing Optimization level	`ROUTER_TIMING_OPTIMIZATION_LEVEL`	`NORMAL, MINIMUM, MAXIMUM`	Global

Related Links

Design Optimization Overview on page 102

9.9.2 Resource Utilization Optimization Techniques

This table lists the `.qsf` file variable name and applicable values for Resource Utilization Optimization settings.

Table 40. Resource Utilization Optimization Settings

Setting Name	.qsf File Variable Name	Values	Type
Auto Packed Registers [1]	AUTO_PACKED_REGISTERS_<device family name>	OFF, NORMAL, MINIMIZE AREA, MINIMIZE AREA WITH CHAINS, AUTO	Global, Instance
Perform WYSIWYG Primitive Resynthesis	ADV_NETLIST_OPT_SYNTH_WYSIWYG_REMAP	ON, OFF	Global, Instance
Optimization Technique	<device family name>_OPTIMIZATION_TECHNIQUE	AREA, SPEED, BALANCED	Global, Instance
Speed Optimization Technique for Clock Domains	SYNTH_CRITICAL_CLOCK	ON, OFF	Instance
State Machine Encoding	STATE_MACHINE_PROCESSING	AUTO, ONE-HOT, GRAY, JOHNSON, MINIMAL BITS, ONE-HOT, SEQUENTIAL, USER-ENCODE	Global, Instance
Auto RAM Replacement	AUTO_RAM_RECOGNITION	ON, OFF	Global, Instance
Auto ROM Replacement	AUTO_ROM_RECOGNITION	ON, OFF	Global, Instance
Auto Shift Register Replacement	AUTO_SHIFT_REGISTER_RECOGNITION	ON, OFF	Global, Instance
Auto Block Replacement	AUTO_DSP_RECOGNITION	ON, OFF	Global, Instance
Number of Processors for Parallel Compilation	NUM_PARALLEL_PROCESSORS	Integer between 1 and 16 inclusive, or ALL	Global

9.9.3 I/O Timing Optimization Techniques

The table lists the `.qsf` file variable name and applicable values for the I/O timing optimization settings.

Table 41. I/O Timing Optimization Settings

Setting Name	.qsf File Variable Name	Values	Type
Optimize IOC Register Placement For Timing	OPTIMIZE_IOC_REGISTER_PLACEMENT_FOR_TIMING	ON, OFF	Global
Fast Input Register	FAST_INPUT_REGISTER	ON, OFF	Instance
Fast Output Register	FAST_OUTPUT_REGISTER	ON, OFF	Instance
Fast Output Enable Register	FAST_OUTPUT_ENABLE_REGISTER	ON, OFF	Instance
Fast OCT Register	FAST_OCT_REGISTER	ON, OFF	Instance

1 Allowed values for this setting depend on the device family that you select.

9.9.4 Register-to-Register Timing Optimization Techniques

The table lists the `.qsf` file variable name and applicable values for the settings described in *Register-to-Register Timing Optimization Techniques*.

Table 42. Register-to-Register Timing Optimization Settings

Setting Name	.qsf File Variable Name	Values	Type
Perform WYSIWYG Primitive Resynthesis	`ADV_NETLIST_OPT_SYNTH_WYSIWYG_REMAP`	`ON, OFF`	Global, Instance
Fitter Seed	`SEED`	*<integer>*	Global
Maximum Fan-Out	`MAX_FANOUT`	*<integer>*	Instance
Manual Logic Duplication	`DUPLICATE_ATOM`	*<node name>*	Instance
Optimize Power during Synthesis	`OPTIMIZE_POWER_DURING_SYNTHESIS`	`NORMAL, OFF EXTRA_EFFORT`	Global
Optimize Power during Fitting	`OPTIMIZE_POWER_DURING_FITTING`	`NORMAL, OFF EXTRA_EFFORT`	Global

Related Links

Register-to-Register Timing Optimization Techniques on page 131
The next stage of design optimization seeks to improve register-to-register (f_{MAX}) timing.

9.10 Document Revision History

Table 43. Document Revision History

Date	Version	Changes
2017.05.08	17.0.0	• Added topic: *Critical Paths*. • Updated *Register-to-Register Timing* and renamed to *Register-to-Register Timing Analysis*. • Renamed topic: *Timing Analysis with the TimeQuest Timing Analyzer* to *Displaying Path Reports with the TimeQuest Timing Analyzer*. • Removed (LUT-Based Devices) remark from topic titles. • Renamed topic: *Optimizing Timing (LUT-Based Devices)* to *Timing Optimization*. • Renamed topic: *Debugging Timing Failures in the TimeQuest Analyzer* to *Displaying Timing Closure Recommendations for Failing Paths*. • Renamed topic: *Improving Register-to-Register Timing Summary* to *Improving Register-to-Register Timing* . • Removed topics: *Tips for Locating Multiple Paths to the Chip Planner*, *LogicLock Assignments* and *Hierarchy Assignments*, . • Removed reference to deprecated Fitter Effort Logic Option. • Removed information about Pin Advisor and Resource Optimization Advisor. • Removed figure: Clock Regions
2016.10.31	16.1.0	• Implemented Intel rebranding.
2016.05.02	16.0.0	• Removed information about deprecated physical synthesis options. • Added information about monitoring clustering difficulty.

continued...

Date	Version	Changes
2015.11.02	15.1.0	• Added: *Periphery to Core Register Placement and Routing Optimization*. • Changed instances of *Quartus II* to *Quartus Prime*.
2014.12.15	14.1.0	• Updated location of Fitter Settings, Analysis & Synthesis Settings, and Physical Synthesis Optimizations to Compiler Settings. • Updated DSE II content.
June 2014	14.0.0	• Dita conversion. • Removed content about obsolete devices that are no longer supported in QII software v14.0: Arria GX, Arria II, Cyclone III, Stratix II, Stratix III. • Replaced Megafunction content with IP core content.
November 2013	13.1.0	• Added Design Evaluation for Timing Closure section. • Removed Optimizing Timing (Macrocell-Based CPLDs) section. • Updated Optimize Multi-Corner Timing and Fitter Aggressive Routability Optimization. • Updated Timing Analysis with the TimeQuest Timing Analyzer to show how to access the **Report All Summaries** command. • Updated Ignored Timing Constraints to include a help link to *Fitter Summary Reports* with the **Ignored Assignment Report** information.
May 2013	13.0.0	• Renamed chapter title from Area and Timing Optimization to Timing Closure and Optimization. • Removed design and area/resources optimization information. • Added the following sections: Fitter Aggressive Routability Optimization. Tips for Analyzing Paths from/to the Source and Destination of Critical Path. Tips for Locating Multiple Paths to the Chip Planner. Tips for Creating a .tcl Script to Monitor Critical Paths Across Compiles.
November 2012	12.1.0	• Updated "Initial Compilation: Optional Fitter Settings" on page 13–2, "I/O Assignments" on page 13–2, "Initial Compilation: Optional Fitter Settings" on page 13–2, "Resource Utilization" on page 13–9, "Routing" on page 13–21, and "Resolving Resource Utilization Problems" on page 13–43.
June 2012	12.0.0	• Updated "Optimize Multi-Corner Timing" on page 13–6, "Resource Utilization" on page 13–10, "Timing Analysis with the TimeQuest Timing Analyzer" on page 13–12, "Using the Resource Optimization Advisor" on page 13–15, "Increase Placement Effort Multiplier" on page 13–22, "Increase Router Effort Multiplier" on page 13–22 and "Debugging Timing Failures in the TimeQuest Analyzer" on page 13–24. • Minor text edits throughout the chapter.
November 2011	11.1.0	• Updated the "Timing Requirement Settings", "Standard Fit", "Fast Fit", "Optimize Multi-Corner Timing", "Timing Analysis with the TimeQuest Timing Analyzer", "Debugging Timing Failures in the TimeQuest Analyzer", "LogicLock Plus Assignments", "Tips for Analyzing Failing Clock Paths that Cross Clock Domains", "Flatten the Hierarchy During Synthesis", "Fast Input, Output, and Output Enable Registers", and "Hierarchy Assignments" sections • Updated Table 13–6 • Added the "Spine Clock Limitations" section • Removed the Change State Machine Encoding section from page 19 • Removed Figure 13-5 • Minor text edits throughout the chapter

continued...

Date	Version	Changes
May 2011	11.0.0	• Reorganized sections in "Initial Compilation: Optional Fitter Settings" section • Added new information to "Resource Utilization" section • Added new information to "Duplicate Logic for Fan-Out Control" section • Added links to Help • Additional edits and updates throughout chapter
December 2010	10.1.0	• Added links to Help • Updated device support • Added "Debugging Timing Failures in the TimeQuest Analyzer" section • Removed Classic Timing Analyzer references • Other updates throughout chapter
August 2010	10.0.1	Corrected link
July 2010	10.0.0	• Moved Compilation Time Optimization Techniques section to new *Reducing Compilation Time* chapter • Removed references to Timing Closure Floorplan • Moved Smart Compilation Setting and Early Timing Estimation sections to new *Reducing Compilation Time* chapter • Added Other Optimization Resources section • Removed outdated information • Changed references to DSE chapter to Help links • Linked to Help where appropriate • Removed Referenced Documents section

Related Links

Altera Documentation Archive

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

10 Power Optimization

10.1 Power Optimization

The Quartus Prime software offers power-driven compilation to fully optimize device power consumption. Power-driven compilation focuses on reducing your design's total power consumption using power-driven synthesis and power-driven place-and-route.

This chapter describes the power-driven compilation feature and flow in detail, as well as low power design techniques that can further reduce power consumption in your design. The techniques primarily target Arria, Stratix, and Cyclone series of devices. These devices utilize a low-k dielectric material that dramatically reduces dynamic power and improves performance. Arria series, Stratix IV, and Stratix V device families include efficient logic structures called adaptive logic modules (ALMs) that obtain maximum performance while minimizing power consumption. Cyclone device families offer the optimal blend of high performance and low power in a low-cost FPGA.

Intel provides the Quartus Prime Power Analyzer to aid you during the design process by delivering fast and accurate estimations of power consumption. You can minimize power consumption, while taking advantage of the industry's leading FPGA performance, by using the tools and techniques described in this chapter.

Total FPGA power consumption is comprised of I/O power, core static power, and core dynamic power. This chapter focuses on design optimization options and techniques that help reduce core dynamic power and I/O power. In addition to these techniques, there are additional power optimization techniques available for specific devices. These techniques include:

- Programmable Power Technology
- Device Speed Grade Selection

Related Links

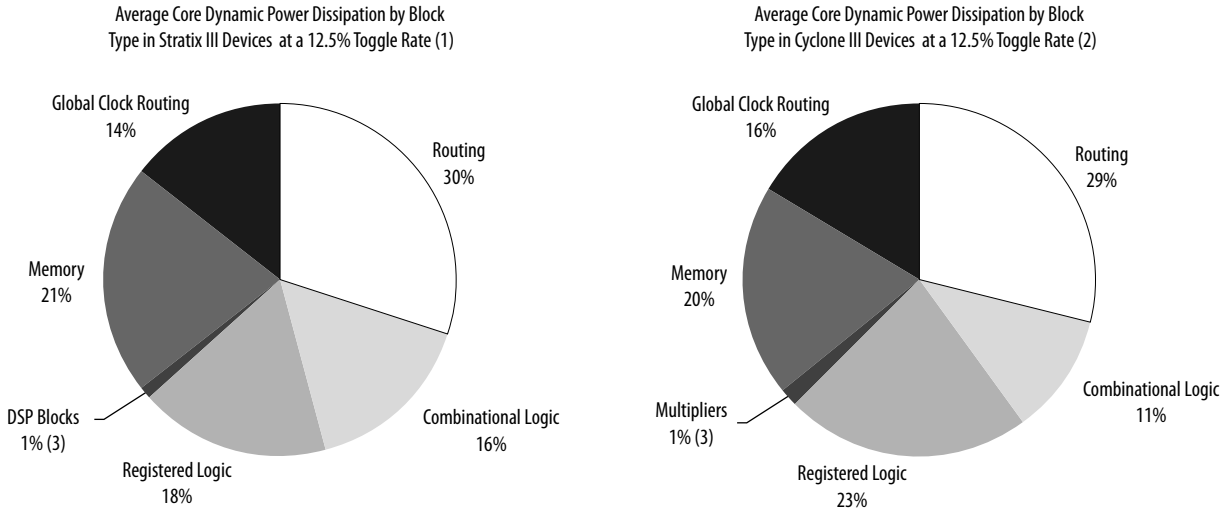
- Literature and Technical Documentation
- Power Analysis
  In *Quartus Prime Pro Edition Handbook Volume 3*
- AN 514: Power Optimization in Stratix IV FPGAs

10.2 Power Dissipation

You can refine techniques that reduce power consumption in your design by understanding the sources of power dissipation.

The following figure shows the power dissipation of Stratix and Cyclone devices in different designs. All designs were analyzed at a fixed clock rate of 100 MHz and exhibited varied logic resource utilization across available resources.

Figure 57. Average Core Dynamic Power Dissipation



Notes:

1. 103 different designs were used to obtain these results.
2. 96 different designs were used to obtain these results.
3. In designs using DSP blocks, DSPs consumed 5% of core dynamic power.

In Stratix and Cyclone device families, a series of column and row interconnect wires of varying lengths provide signal interconnections between logic array blocks (LABs), memory block structures, and digital signal processing (DSP) blocks or multiplier blocks. These interconnects dissipate the largest component of device power.

FPGA combinational logic is another source of power consumption. The basic building block of logic in the latest Stratix series devices is the ALM, and in Cyclone IV GX devices, it is the logic element (LE).

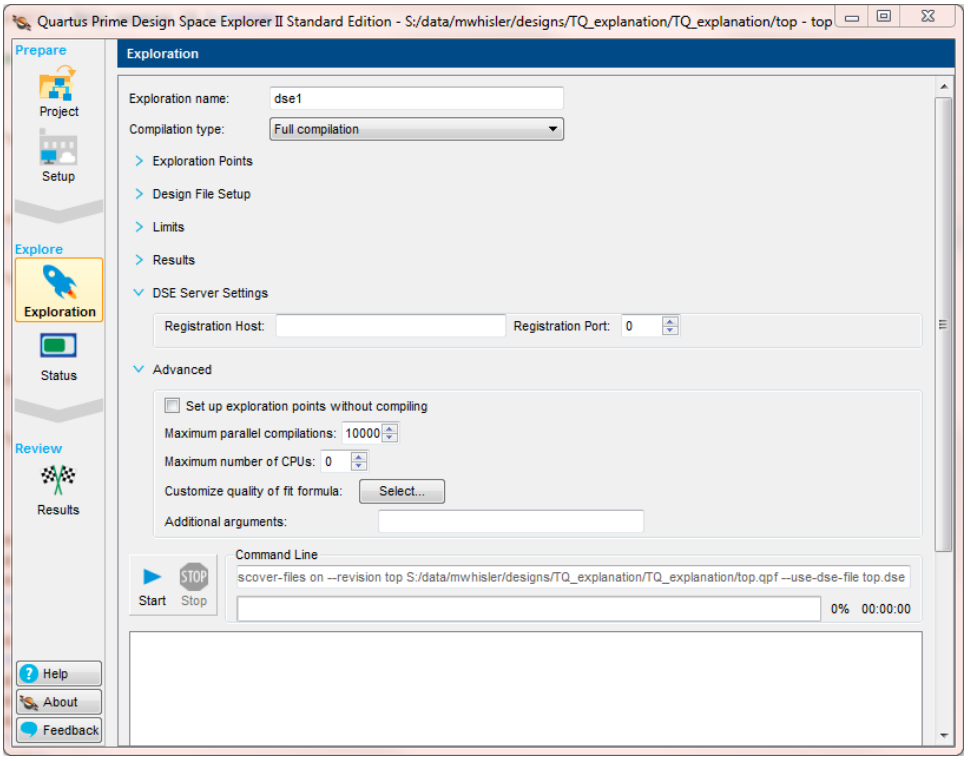
For more information about ALMs and LEs in Cyclone or Stratix devices, refer to the respective device handbook.

Memory and clock resources are other major consumers of power in FPGAs. Stratix devices feature the TriMatrix memory architecture. TriMatrix memory includes 512-bit M512 blocks, 4-Kbit M4K blocks, and 512-Kbit M-RAM blocks, which are configurable to support many features. Stratix IV TriMatrix on-chip memory is an enhancement based upon the Stratix II FPGA TriMatrix memory and includes three sizes of memory blocks: MLAB blocks, M9K blocks, and M144K blocks. Stratix IV and Stratix V devices feature Programmable Power Technology, an advanced architecture that enables a smooth trade-off between speed and power. The core of each Stratix IV and Stratix V device is divided into tiles, each of which may be put into a high-speed or low-power mode. The primary benefit of Programmable Power Technology is to reduce static power, with a secondary benefit being a small reduction in dynamic power. Cyclone IV GX devices have 9-Kbit M9K memory blocks.

10.3 Design Space Explorer II

Design Space Explorer II (DSE) is an easy-to-use, self-guided design optimization utility that is included in the Quartus Prime software. DSE II explores and reports optimal Quartus Prime software options for your design, targeting either power optimization, design performance, or area utilization improvements. You can use DSE II to implement the techniques described in this chapter.

Figure 58. Design Space Explorer II User Interface



The power optimizations, under **Exploration mode**, target overall design power improvements. These settings focus on applying different options that specifically reduce total design thermal power.

By default, the Quartus Prime Power Analyzer is run for every exploration performed by DSE II when power optimizations are selected. This helps you debug your design and determine trade-offs between power requirements and performance optimization.

DSE II automatically tries different combinations of netlist optimizations and advanced Quartus Prime software compiler settings, and reports the best settings for your design, based on your chosen primary optimization goal. You can try different seeds with DSE II if you are fairly close to meeting your timing or area requirements and find one seed that meets timing or area requirements. Finally, DSE II can run compilations on a remote compute farm, which shortens the timing closure process

- Name your DSE II session and specify the type of compilation to perform.
- Set **Exploration Points** and specify Exploration mode and the number and types of **Seeds** to use.
- Specify the **Design File Setup** including the use of a specified Quartus Archive File (`.qar`) or create a new one.
- Specify **Limits** to the operation of DSE II.
- Specify the type of **Results** to save.
- When using a remote compute farm, DSE II uses the values in the **DSE Server Settings** box to specify a registration host and network ports to connect.
- Options in the **Advanced** settings allow you to specify options such as:
  - Turn on the option to specify exploration points without compiling.
  - Specify the **Maximum number of parallel compilations** used by DSE II.
  - Specify the **Maximum number of CPUs** that can be used by DSE II.
  - Specify a quality of fit formula.

When you have completed your configuration, you can perform an exploration by clicking **Start**.

Related Links

- Optimizing with Design Space Explorer II
  In *Quartus Prime Pro Edition Handbook Volume 1*
- Launch Design Space Explorer Command (Tools Menu)
  in Quartus Prime Help

10.4 Power-Driven Compilation

The standard Quartus Prime compilation flow consists of Analysis and Synthesis, placement and routing, Assembly, and Timing Analysis. Power-driven compilation takes place at the Analysis and Synthesis and Place-and-Route stages.

Quartus Prime software settings that control power-driven compilation are located in the **power optimization during synthesis** list in the **Advanced Settings (Synthesis)** dialog box, and the **power optimization during fitting** list on the **Advanced Fitter Settings** dialog box. The following sections describes these power optimization options at the Analysis and Synthesis and Fitter levels.

10.4.1 Power-Driven Synthesis

Synthesis netlist optimization occurs during the synthesis stage of the compilation flow. The optimization technique makes changes to the synthesis netlist to optimize your design according to the selection of area, speed, or power optimization. This section describes power optimization techniques at the synthesis level.

To access the Power Optimization During Synthesis option, click **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Synthesis).**

You can apply these settings on a project or entity level.

Table 44. Optimize Power During Synthesis Options

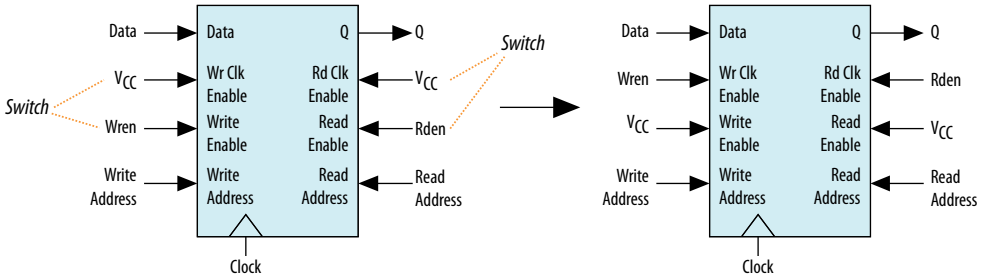
Settings	Description
Off	No netlist, placement, or routing optimizations are performed to minimize power.
Normal compilation (Default)	Low compute effort algorithms are applied to minimize power through netlist optimizations as long as they are not expected to reduce design performance.
Extra effort	High compute effort algorithms are applied to minimize power through netlist optimizations. Max performance might be impacted.

The **Normal compilation** setting is turned on by default. This setting performs memory optimization and power-aware logic mapping during synthesis.

Memory blocks can represent a large fraction of total design dynamic power. Minimizing the number of memory blocks accessed during each clock cycle can significantly reduce memory power. Memory optimization involves effective movement of user-defined read/write enable signals to associated read-and-write clock enable signals for all memory types.

A default implementation of a simple dual-port memory block in which write-clock enable signals and read-clock enable signals are connected to V_{CC}, making both read and write memory ports active during each clock cycle.

Figure 59. Memory Transformation



Memory transformation effectively moves the read-enable and write-enable signals to the respective read-clock enable and write-clock enable signals. By using this technique, memory ports are shut down when they are not accessed. This significantly reduces your design's memory power consumption. For Stratix IV and Stratix V devices, the memory transformation takes place at the Fitter level by selecting the **Normal compilation** settings for the power optimization option.

In Cyclone IV GX and StratixIV devices, the specified read-during-write behavior can significantly impact the power of single-port and bidirectional dual-port RAMs. It is best to set the read-during-write parameter to "Don't care" (at the HDL level), as it allows an optimization whereby the `read-enable` signal can be set to the inversion of the existing `write-enable` signal (if one exists). This allows the core of the RAM to shut down (that is, not toggle), which saves a significant amount of power.

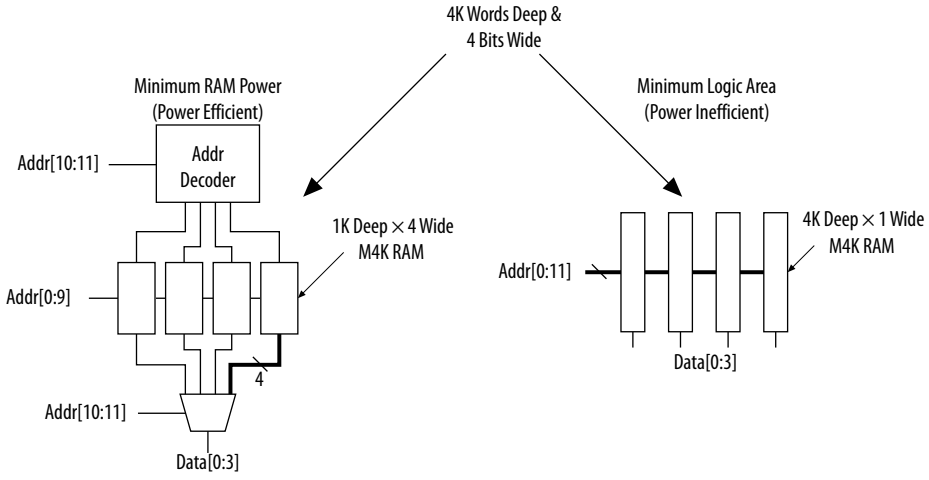
The other type of power optimization that takes place with the **Normal compilation** setting is power-aware logic mapping. The power-aware logic mapping reduces power by rearranging the logic during synthesis to eliminate nets with high toggle rates.

The **Extra effort** setting performs the functions of the **Normal compilation** setting and other memory optimizations to further reduce memory power by shutting down memory blocks that are not accessed. This level of memory optimization can require extra logic, which can reduce design performance.

The **Extra effort** setting also performs power-aware memory balancing. Power-aware memory balancing automatically chooses the best memory configuration for your memory implementation and provides optimal power saving by determining the number of memory blocks, decoder, and multiplexer circuits required. If you have not previously specified target-embedded memory blocks for your design's memory functions, the power-aware balancer automatically selects them during memory implementation.

The following figure is an example of a 4k × 4 (4k deep and 4 bits wide) memory implementation in two different configurations using M4K memory blocks available in some Stratix devices.

Figure 60. 4K × 4 Memory Implementation Using Multiple M4K Blocks



The minimum logic area implementation uses M4K blocks configured as 4k × 1. This implementation is the default in the Quartus Prime software because it has the minimum logic area (0 logic cells) and the highest speed. However, all four M4K blocks are active on each memory access in this implementation, which increases RAM power. The minimum RAM power implementation is created by selecting **Extra effort** in the **power optimization** list. This implementation automatically uses four M4K blocks configured as 1k × 4 for optimal power saving. An address decoder is implemented by the RAM megafunction to select which of the four M4K blocks should be activated on a given cycle, based on the state of the top two user address bits. The RAM megafunction automatically implements a multiplexer to feed the downstream logic by choosing the appropriate M4K output. This implementation reduces RAM power because only one M4K block is active on any cycle, but it requires extra logic cells, costing logic area and potentially impacting design performance.

There is a trade-off between power saved by accessing fewer memories and power consumed by the extra decoder and multiplexor logic. The Quartus Prime software automatically balances the power savings against the costs to choose the lowest power configuration for each logical RAM. The benchmark data shows that the power-driven synthesis can reduce memory power consumption by as much as 60% in Stratix devices.

Memory optimization options can also be controlled by the Low_Power_Mode parameter in the **Default Parameters** page of the **Settings** dialog box. The settings for this parameter are **None**, **Auto**, and **ALL**. **None** corresponds to the **Off** setting in the **power optimization** list. **Auto** corresponds to the **Normal compilation** setting and **ALL** corresponds to the **Extra effort** setting, respectively. You can apply power optimization either on a compiler basis or on individual entities. The Low_Power_Mode parameter always takes precedence over the **Optimize Power for Synthesis** option for power optimization on memory.

You can also set the MAXIMUM_DEPTH parameter manually to configure the memory for low power optimization. This technique is the same as the power-aware memory balancer, but it is manual rather than automatic like the **Extra effort** setting in the **power optimization** list. You can set the MAXIMUM_DEPTH parameter for memory modules manually in the megafunction instantiation or in the IP Catalog for power optimization. The MAXIMUM_DEPTH parameter always takes precedence over the **Optimize Power for Synthesis** options for power optimization on memory optimization.

Related Links

Reducing Memory Power Consumption on page 168
The memory blocks in FPGA devices can represent a large fraction of typical core dynamic power.

10.4.2 Power-Driven Fitter

The **Compiler Settings** page provides access to **power optimization** settings.

You can apply these settings only on a project-wide basis. The **Extra effort** setting for the Fitter requires extensive effort to optimize the design for power and can increase the compilation time.

Table 45. Power-Driven Fitter Option

Settings	Description
Off	No netlist, placement, or routing optimizations are performed to minimize power.
Normal compilation (Default)	Low compute effort algorithms are applied to minimize power through placement and routing optimizations as long as they are not expected to reduce design performance.
Extra effort	High compute effort algorithms are applied to minimize power through placement and routing optimizations. Max performance might be impacted.

The **Normal compilation** setting is selected by default and performs DSP optimization by creating power-efficient DSP block configurations for your DSP functions. For Stratix IV and Stratix V devices, this setting, which is based on timing constraints entered for the design, enables the Programmable Power Technology to configure tiles as high-speed mode or low-power mode. Programmable Power Technology is always turned **ON** even when the **OFF** setting is selected for the **power optimization** option. Tiles are the combination of LAB and MLAB pairs (including the adjacent routing associated with LAB and MLAB), which can be configured to operate in high-speed or low-power mode. This level of power optimization does not have any affect on the fitting, timing results, or compile time.

The **Extra effort** setting performs the functions of the **Normal compilation** setting and other place-and-route optimizations during fitting to fully optimize the design for power. The Fitter applies an extra effort to minimize power even after timing

requirements have been met by effectively moving the logic closer during placement to localize high-toggling nets, and using routes with low capacitance. However, this effort can increase the compilation time.

The **Extra effort** setting uses a Value Change Dump File (`.vcd`) that guides the Fitter to fully optimize the design for power, based on the signal activity of the design. The best power optimization during fitting results from using the most accurate signal activity information. If you do not have a `.vcd` file, the Quartus Prime software uses assignments, clock assignments, and vectorless estimation values (Power Analyzer Tool settings) to estimate the signal activities. This information is used to optimize your design for power during fitting. The benchmark data shows that the power-driven Fitter technique can reduce power consumption by as much as 19% in Stratix devices. On average, you can reduce core dynamic power by 16% with the Extra effort synthesis and Extra effort fitting settings, as compared to the **Off** settings in both synthesis and Fitter options for power-driven compilation.

Note: Only the **Extra effort** setting in the **power optimization** list for the Fitter option uses the signal activities (from `.vcd` files) during fitting. The settings made in the **Power Analyzer Settings** page in the **Settings** dialog box are used to calculate the signal activity of your design.

Related Links

- AN 514: Power Optimization in Stratix IV FPGAs
- Power Analysis
  In *Quartus Prime Pro Edition Handbook Volume 3*

10.4.3 Area-Driven Synthesis

Using area optimization rather than timing or delay optimization during synthesis saves power because you use fewer logic blocks. Using less logic usually means less switching activity.

The Quartus Prime software provides **Speed**, **Balanced**, or **Area** for the **Optimization Technique** technique option. You can also specify this logic option for specific modules in your design with the Assignment Editor in cases where you want to reduce area using the **Area** setting (potentially at the expense of register-to-register timing performance) while leaving the default **Optimization Technique** setting at **Balanced** (for the best trade-off between area and speed for certain device families). The **Speed Optimization Technique** can increase the resource usage of your design if the constraints are too aggressive, and can also result in increased power consumption.

The benchmark data shows that the area-driven technique can reduce power consumption by as much as 31% in Stratix devices and as much as 15% in Cyclone devices.

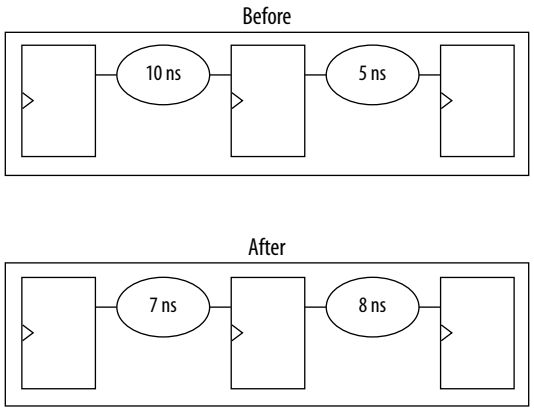
10.4.4 Gate-Level Register Retiming

You can also use gate-level register retiming to reduce circuit switching activity. Retiming shuffles registers across combinational blocks without changing design functionality.

The **Perform gate-level register retiming** option in the Quartus Prime software enables the movement of registers across combinational logic to balance timing, allowing the software to trade off the delay between critical and noncritical paths.

Retiming uses fewer registers than pipelining. In this example of gate-level register retiming, the 10 ns critical delay is reduced by moving the register relative to the combinational logic, resulting in the reduction of data depth and switching activity.

Figure 61. Gate-Level Register Retiming



Gate-level register retiming makes changes at the gate level. If you are using an atom netlist from a third-party synthesis tool, you must also select the **Perform WYSIWYG primitive resynthesis** option to undo the atom primitives to gates mapping (so that register retiming can be performed), and then to remap gates to Intel primitives.

Related Links

Netlist Optimizations and Physical Synthesis on page 215

10.5 Design Guidelines

Several low-power design techniques can reduce power consumption when applied during FPGA design implementation. This section provides detailed design techniques for Cyclone IV GX devices that affect overall design power. The results of these techniques might be different from design to design.

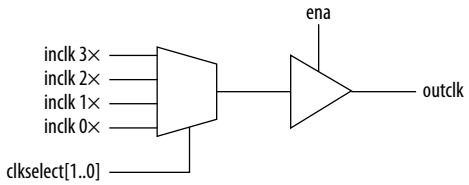
10.5.1 Clock Power Management

Clocks represent a significant portion of dynamic power consumption due to their high switching activity and long paths. Actual clock-related power consumption is higher than this because the power consumed by local clock distribution within logic, memory, and DSP or multiplier blocks is included in the power consumption for the respective blocks.

Clock routing power is automatically optimized by the Quartus Prime software, which enables only those portions of the clock network that are required to feed downstream registers. Power can be further reduced by gating clocks when they are not required. It is possible to build clock-gating logic, but this approach is not recommended because it is difficult to generate a glitch free clock in FPGAs using ALMs or LEs.

Cyclone IV , Stratix IV, and Stratix V devices use clock control blocks that include an enable signal. A clock control block is a clock buffer that lets you dynamically enable or disable the clock network and dynamically switch between multiple sources to drive the clock network. You can use the Quartus Prime IP Catalog to create this clock control block with the ALTCLKCTRL megafunction. Cyclone IV , Stratix IV, and Stratix V devices provide clock control blocks for global clock networks. In addition, Stratix IV, and Stratix V devices have clock control blocks for regional clock networks. The dynamic clock enable feature lets internal logic control the clock network. When a clock network is powered down, all the logic fed by that clock network does not toggle, thereby reducing the overall power consumption of the device. For example, the following shows a 4-input clock control block diagram.

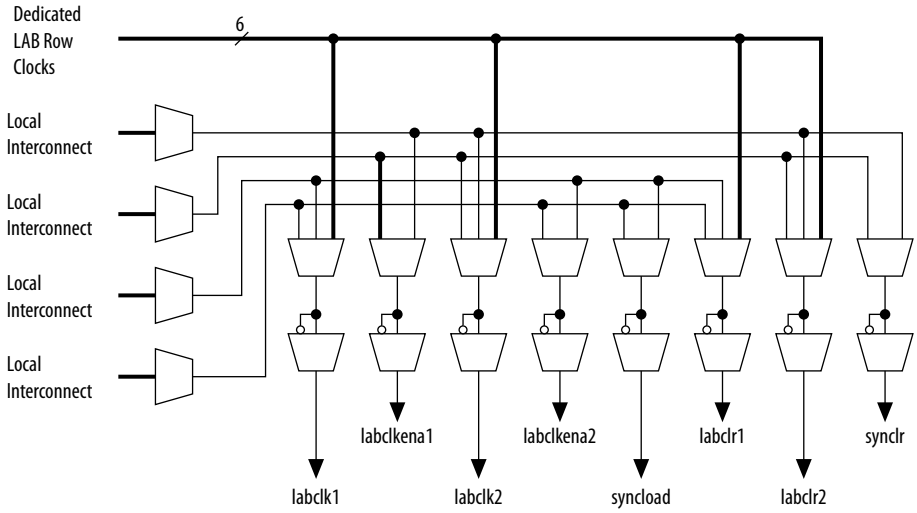
Figure 62. Clock Control Block Diagram



The enable signal is applied to the clock signal before being distributed to global routing. Therefore, the enable signal can either have a significant timing slack (at least as large as the global routing delay) or it can reduce the f_{MAX} of the clock signal.

Another contributor to clock power consumption is the LAB clock that distributes a clock to the registers within a LAB. LAB clock power can be the dominant contributor to overall clock power. For example, in Cyclone devices, each LAB can use two clocks and two clock enable signals, as shown in the following figure. Each LAB's clock signal and clock enable signal are linked. For example, an LE in a particular LAB using the `labclk1` signal also uses the `labclkena1` signal.

Figure 63. LAB-Wide Control Signals



To reduce LAB-wide clock power consumption without disabling the entire clock tree, use the LAB-wide clock enable to gate the LAB-wide clock. The Quartus Prime software automatically promotes register-level clock enable signals to the LAB-level. All

registers within an LAB that share a common clock and clock enable are controlled by a shared gated clock. To take advantage of these clock enables, use a clock enable construct in the relevant HDL code for the registered logic.

Related Links

Clock Control Block Megafunction User Guide (ALTCLKCTRL)

10.5.1.1 LAB-Wide Clock Enable Example

This VHDL code makes use of a LAB-wide clock enable. This clock-gating logic is automatically turned into an LAB-level clock enable signal.

```
IF clk'event AND clock = '1' THEN
    IF logic_is_enabled = '1' THEN
        reg <= value;
    ELSE
        reg <= reg;
    END IF;
END IF;
```

10.5.1.2 Reducing Memory Power Consumption

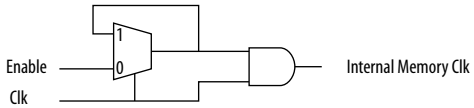
The memory blocks in FPGA devices can represent a large fraction of typical core dynamic power.Memory consumes approximately 20% of the core dynamic power in typical some device designs. Memory blocks are unlike most other blocks in the device because most of their power is tied to the clock rate, and is insensitive to the toggle rate on the data and address lines.

When a memory block is clocked, there is a sequence of timed events that occur within the block to execute a read or write. The circuitry controlled by the clock consumes the same amount of power regardless of whether or not the address or data has changed from one cycle to the next. Thus, the toggle rate of input data and the address bus have no impact on memory power consumption.

The key to reducing memory power consumption is to reduce the number of memory clocking events. You can achieve this through clock network-wide gating, or on a per-memory basis through use of the clock enable signals on the memory ports.

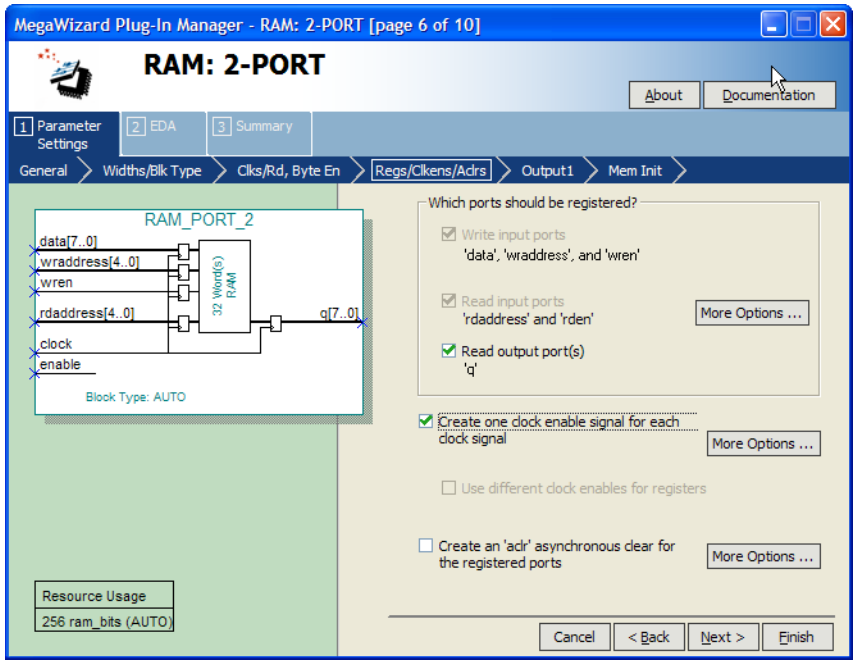
The logical view of the internal clock of the memory block. Use the appropriate enable signals on the memory to make use of the clock enable signal instead of gating the clock.

Figure 64. Memory Clock Enable Signal



Using the clock enable signal enables the memory only when necessary and shuts it down for the rest of the time, reducing the overall memory power consumption. You can create these enable signals by selecting the **Clock enable signal** option for the appropriate port when generating the memory block function.

Figure 65. RAM 2-Port Clock Enable Signal Selectable Option

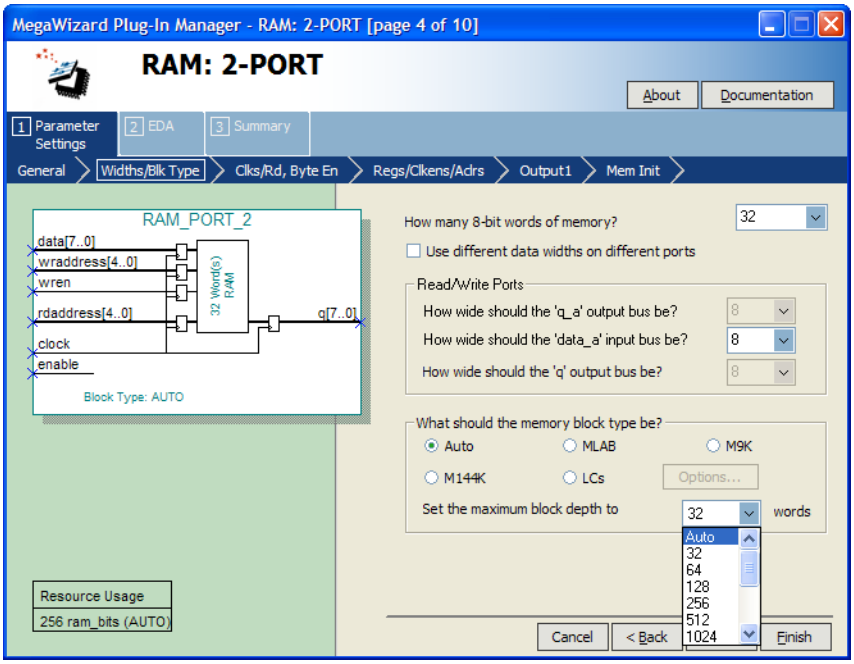


For example, consider a design that contains a 32-bit-wide M4K memory block in ROM mode that is running at 200 MHz. Assuming that the output of this block is only required approximately every four cycles, this memory block will consume 8.45 mW of dynamic power according to the demands of the downstream logic. By adding a small amount of control logic to generate a read clock enable signal for the memory block only on the relevant cycles, the power can be cut 75% to 2.15 mW.

You can also use the MAXIMUM_DEPTH parameter in your memory megafunction to save power in Cyclone IV GX, Stratix IV, and Stratix V devices; however, this approach might increase the number of LEs required to implement the memory and affect design performance.

You can set the MAXIMUM_DEPTH parameter for memory modules manually in the megafunction instantiation. The Quartus Prime software automatically chooses the best design memory configuration for optimal power.

Figure 66. RAM 2-Port Maximum Depth Selectable Option



Related Links

- Power-Driven Compilation on page 161
  The standard Quartus Prime compilation flow consists of Analysis and Synthesis, placement and routing, Assembly, and Timing Analysis. Power-driven compilation takes place at the Analysis and Synthesis and Place-and-Route stages.
- Clock Power Management on page 166
  Clocks represent a significant portion of dynamic power consumption due to their high switching activity and long paths.

10.5.1.3 Memory Power Reduction Example

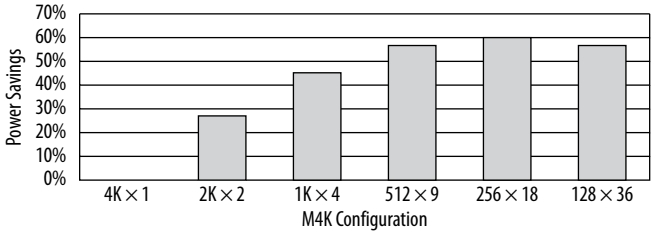
Power usage measurements for a 4K × 36 simple dual-port memory implemented using multiple M4K blocks in a Stratix device. For each implementation, the M4K blocks are configured with a different memory depth.

Table 46. 4K × 36 Simple Dual-Port Memory Implemented Using Multiple M4K Blocks

M4K Configuration	Number of M4K Blocks	ALUTs
4K × 1 (Default setting)	36	0
2K × 2	36	40
1K × 4	36	62
512 × 9	32	143
256 × 18	32	302
128 × 36	32	633

Using the MAXIMUM_DEPTH parameter can save power. For all implementations, a user-provided read enable signal is present to indicate when read data is required. Using this power-saving technique can reduce power consumption by as much as 60%.

Figure 67. Power Savings Using the MAXIMUM_DEPTH Parameter



As the memory depth becomes more shallow, memory dynamic power decreases because unaddressed M4K blocks can be shut off using a decoded combination of address bits and the read enable signal. For a 128-deep memory block, power used by the extra LEs starts to outweigh the power gain achieved by using a more shallow memory block depth. The power consumption of the memory blocks and associated LEs depends on the memory configuration.

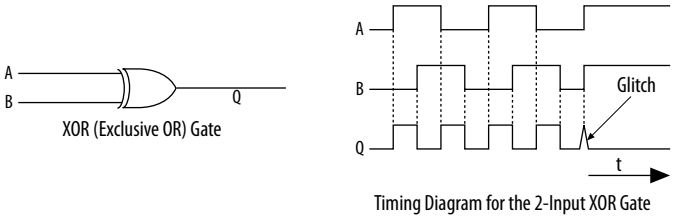
Note: The SOPC Builder and Qsys Pro system do not offer specific power savings control for on-chip memory block. There is no read enable, write enable, or clock enable that you can enable in the on-chip RAM megafunction to shut down the RAM block in the SOPC Builder and Qsys Pro system.

10.5.2 Pipelining and Retiming

Designs with many glitches consume more power because of faster switching activity. Glitches cause unnecessary and unpredictable temporary logic switches at the output of combinational logic. A glitch usually occurs when there is a mismatch in input signal timing leading to unequal propagation delay.

For example, consider an input change on one input of a 2-input XOR gate from 1 to 0, followed a few moments later by an input change from 0 to 1 on the other input. For a moment, both inputs become 1 (high) during the state transition, resulting in 0 (low) at the output of the XOR gate. Subsequently, when the second input transition takes place, the XOR gate output becomes 1 (high). During signal transition, a glitch is produced before the output becomes stable.

Figure 68. XOR Gate Showing Glitch at the Output

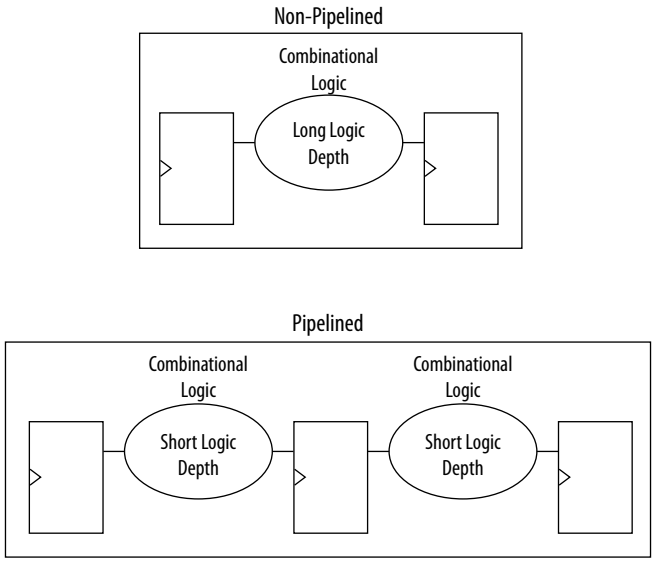


This glitch can propagate to subsequent logic and create unnecessary switching activity, increasing power consumption. Circuits with many XOR functions, such as arithmetic circuits or cyclic redundancy check (CRC) circuits, tend to have many glitches if there are several levels of combinational logic between registers.

Pipelining can reduce design glitches by inserting flipflops into long combinational paths. Flipflops do not allow glitches to propagate through combinational paths. Therefore, a pipelined circuit tends to have less glitching. Pipelining has the additional benefit of generally allowing higher clock speed operations, although it does increase the latency of a circuit (in terms of the number of clock cycles to a first result).

An example where pipelining is applied to break up a long combinational path.

Figure 69. Pipelining Example



Pipelining is very effective for glitch-prone arithmetic systems because it reduces switching activity, resulting in reduced power dissipation in combinational logic. Additionally, pipelining allows higher-speed operation by reducing logic-level numbers between registers. The disadvantage of this technique is that if there are not many glitches in your design, pipelining can increase power consumption by adding unnecessary registers. Pipelining can also increase resource utilization. The benchmark data shows that pipelining can reduce dynamic power consumption by as much as 30% in Cyclone and Stratix devices.

10.5.3 Architectural Optimization

You can use design-level architectural optimization by taking advantage of specific device architecture features.These features include dedicated memory and DSP or multiplier blocks available in FPGA devices to perform memory or arithmetic-related functions. You can use these blocks in place of LUTs to reduce power consumption. For example, you can build large shift registers from RAM-based FIFO buffers instead of building the shift registers from the LE registers.

The Stratix device family allows you to efficiently target small, medium, and large memories with the TriMatrix memory architecture. Each TriMatrix memory block is optimized for a specific function. M512 memory blocks are more power-efficient than the distributed memory structures in some competing FPGAs. The M4K memory blocks are used to implement buffers for a wide variety of applications, including processor code storage, large look-up table implementation, and large memory applications. The

M-RAM blocks are useful in applications where a large volume of data must be stored on-chip. Effective utilization of these memory blocks can have a significant impact on power reduction in your design.

The latest Stratix and Cyclone device families have configurable M9K memory blocks that provide various memory functions such as RAM, FIFO buffers, and ROM.

Related Links

Timing Closure and Optimization on page 116

10.5.4 I/O Power Guidelines

Nonterminated I/O standards such as LVTTL and LVCMOS have a rail-to-rail output swing. The voltage difference between logic-high and logic-low signals at the output pin is equal to the V_{CCIO} supply voltage. If the capacitive loading at the output pin is known, the dynamic power consumed in the I/O buffer can be calculated.

$P = 0.5 \times F \times C \times V^2$

In this equation, F is the output transition frequency and C is the total load capacitance being switched. V is equal to V_{CCIO} supply voltage. Because of the quadratic dependence on V_{CCIO}, lower voltage standards consume significantly less dynamic power.

Transistor-to-transistor logic (TTL) I/O buffers consume very little static power. As a result, the total power consumed by a LVTTL or LVCMOS output is highly dependent on load and switching frequency.

When using resistively terminated I/O standards like SSTL and HSTL, the output load voltage swings by a small amount around some bias point. The same dynamic power equation is used, where V is the actual load voltage swing. Because this is much smaller than V_{CCIO}, dynamic power is lower than for nonterminated I/O under similar conditions. These resistively terminated I/O standards dissipate significant static (frequency-independent) power, because the I/O buffer is constantly driving current into the resistive termination network. However, the lower dynamic power of these I/O standards means they often have lower total power than LVCMOS or LVTTL for high-frequency applications. Use the lowest drive strength I/O setting that meets your speed and waveform requirements to minimize I/O power when using resistively terminated standards.

You can save a small amount of static power by connecting unused I/O banks to the lowest possible V_{CCIO} voltage of 1.2 V.

When calculating I/O power, the Power Analyzer uses the default capacitive load set for the I/O standard in the **Capacitive Loading** page of the **Device and Pin Options** dialog box. Any other components defined in the board trace model are not taken into account for the power measurement.

For Cyclone IV GX, Stratix IV, and Stratix V devices, Advanced I/O Timing is always used, which uses the full board trace model.

Related Links

- Managing Device I/O Pins on page 18
- Stratix Series FPGA I/O Connectivity
- I/O Features in Stratix IV Devices

- I/O Features in Cyclone IV Devices

10.5.5 Dynamically Controlled On-Chip Terminations

Stratix IV and Stratix V FPGAs offer dynamic on-chip termination (OCT). Dynamic OCT enables series termination (RS) and parallel termination (RT) to dynamically turn on/off during the data transfer. This feature is especially useful when Stratix IV and Stratix V FPGAs are used with external memory interfaces, such as interfacing with DDR memories.

Compared to conventional termination, dynamic OCT reduces power consumption significantly as it eliminates the constant DC power consumed by parallel termination when transmitting data. Parallel termination is extremely useful for applications that interface with external memories where I/O standards, such as HSTL and SSTL, are used. Parallel termination supports dynamic OCT, which is useful for bidirectional interfaces.

The following is an example of power saving for a DDR3 interface using on-chip parallel termination.

The static current consumed by parallel OCT is equal to the V_{CCIO} voltage divided by 100 W . For DDR3 interfaces that use SSTL-15, the static current is 1.5 V/100 W = 15 mA per pin. Therefore, the static power is 1.5 V ×15 mA = 22.5 mW. For an interface with 72 DQ and 18 DQS pins, the static power is 90 pins × 22.5 mW = 2.025 W. Dynamic parallel OCT disables parallel termination during write operations, so if writing occurs 50% of the time, the power saved by dynamic parallel OCT is 50% × 2.025 W = 1.0125 W.

For more information about dynamic OCT in Stratix IV devices, refer to the chapter in the *Stratix IV Device Handbook*.

Related Links

Stratix IV Device I/O Features
In *Stratix IV Device Handbook*

10.5.6 Power Optimization Advisor

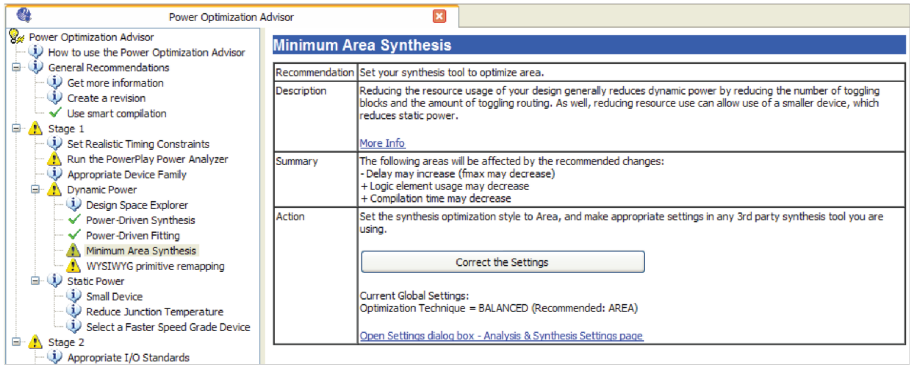
The Quartus Prime software includes the Power Optimization Advisor, which provides specific power optimization advice and recommendations based on the current design project settings and assignments. The advisor covers many of the suggestions listed in this chapter. The following example shows how to reduce your design power with the Power Optimization Advisor.

10.5.6.1 Power Optimization Advisor Example

After compiling your design, run the Power Analyzer to determine your design power and to see where power is dissipated in your design. Based on this information, you can run the Power Optimization Advisor to implement recommendations that can reduce design power.

The Power Optimization Advisor after compiling a design that is not fully optimized for power.

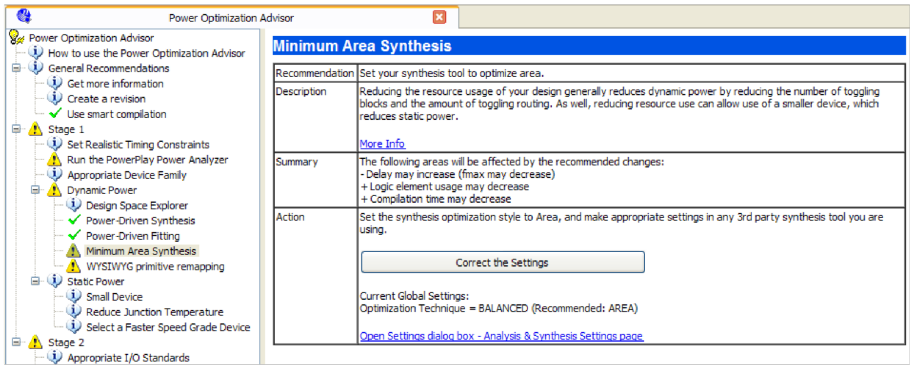
Figure 70. Power Optimization Advisor



The Power Optimization Advisor shows the recommendations that can reduce power in your design. The recommendations are split into stages to show the order in which you should apply the recommended settings. The first stage shows mostly CAD setting options that are easy to implement and highly effective in reducing design power. An icon indicates whether each recommended setting is made in the current project. The checkmark icons for Stage 1 shows the recommendations that are already implemented. The warning icons indicate recommendations that are not followed for this compilation. The information icon shows the general suggestions. Each recommendation includes the description, summary of the effect of the recommendation, and the action required to make the appropriate setting.

There is a link from each recommendation to the appropriate location in the Quartus Prime user interface where you can change the setting. After making the recommended changes, recompile your design. The Power Optimization Advisor indicates with green check marks that the recommendations were implemented successfully. You can use the Power Analyzer to verify your design power results.

Figure 71. Implementation of Power Optimization Advisor Recommendations



The recommendations listed in Stage 2 generally involve design changes, rather than CAD settings changes as in Stage 1. You can use these recommendations to further reduce your design power consumption. Intel recommends that you implement Stage 1 recommendations first, then the Stage 2 recommendations.

10.6 Document Revision History

Table 47. Document Revision History

Date	Version	Changes
2016.10.31	16.1.0	• Implemented Intel rebranding. • Removed statement of support for gate-level timing simulation.
2015.11.02	15.1.0	Changed instances of *Quartus II* to *Quartus Prime*. • Updated screenshot for DSE II GUI. • Added information about remote hosts for DSE II.
2015.05.04	15.0.0	
2014.12.15	14.1.0	• Updated location of Fitter Settings, Analysis & Synthesis Settings, and Physical Synthesis Optimizations to Compiler Settings. • Updated DSE II GUI and optimization settings.
2014.06.30	14.0.0	Updated the format.
May 2013	13.0.0	Added a note to "Memory Power Reduction Example" on Qsys and SOPC Builder power savings limitation for on-chip memory block.
June 2012	12.0.0	Removed survey link.
November 2011	10.0.2	Template update.
December 2010	10.0.1	Template update.
July 2010	10.0.0	• Was chapter 11 in the 9.1.0 release • Updated Figures 14-2, 14-3, 14-6, 14-18, 14-19, and 14-20 • Updated device support • Minor editorial updates
November 2009	9.1.0	• Updated Figure 11-1 and associated references • Updated device support • Minor editorial update
March 2009	9.0.0	• Was chapter 9 in the 8.1.0 release • Updated for the Quartus II software release • Added benchmark results • Removed several sections • Updated Figure 13–1, Figure 13–17, and Figure 13–18
November 2008	8.1.0	• Changed to 8½" × 11" page size • Changed references to altsyncram to RAM • Minor editorial updates
May 2008	8.0.0	• Added support for Stratix IV devices • Updated Table 9–1 and 9–9 • Updated "Architectural Optimization" on page 9–22 • Added "Dynamically-Controlled On-Chip Terminations" on page 9–26 • Updated "Referenced Documents" on page 9–29 • Updated references

Related Links

Altera Documentation Archive

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

11 Area Optimization

This chapter describes techniques to reduce resource usage when designing for Intel devices.

11.1 Resource Utilization

Determining device utilization is important regardless of whether your design achieved a successful fit. If your compilation results in a no-fit error, resource utilization information is important for analyzing the fitting problems in your design.If your fitting is successful, review the resource utilization information to determine whether the future addition of extra logic or other design changes might introduce fitting difficulties. Also, review the resource utilization information to determine if it is impacting timing performance.

To determine resource usage, refer to the **Flow Summary** section of the Compilation Report. This section reports resource utilization, including pins, memory bits, digital signal processing (DSP) blocks, and phase-locked loops (PLLs). **Flow Summary** indicates whether your design exceeds the available device resources. More detailed information is available by viewing the reports under **Resource Section** in the **Fitter** section of the Compilation Report.

Flow Summary shows the overall logic utilization. The Fitter can spread logic throughout the device, which may lead to higher overall utilization.

As the device fills up, the Fitter automatically searches for logic functions with common inputs to place in one ALM. The number of packed registers also increases. Therefore, a design that has high overall utilization might still have space for extra logic if the logic and registers can be packed together more tightly.

The reports under the **Resource Section** in the **Fitter** section of the Compilation Report provide more detailed resource information. The Fitter Resource Usage Summary report breaks down the logic utilization information and provides other resource information, including the number of bits in each type of memory block. This panel also contains a summary of the usage of global clocks, PLLs, DSP blocks, and other device-specific resources.

You can also view reports describing some of the optimizations that occurred during compilation. For example, if you use Quartus Prime synthesis, the reports in the Optimization Results folder in the **Analysis & Synthesis** section include information about registers removed during synthesis. Use this report to estimate device resource utilization for a partial design to ensure that registers were not removed due to missing connections with other parts of the design.

If a specific resource usage is reported as less than 100% and a successful fit cannot be achieved, either there are not enough routing resources or some assignments are illegal. In either case, a message appears in the **Processing** tab of the **Messages** window describing the problem.

If the Fitter finishes unsuccessfully and runs much faster than on similar designs, a resource might be over-utilized or there might be an illegal assignment. If the Quartus Prime software seems to run for an excessively long time compared to runs on similar designs, a legal placement or route probably cannot be found. In the Compilation Report, look for errors and warnings that indicate these types of problems.

You can use the Chip Planner to find areas of the device that have routing congestion on specific types of routing resources. If you find areas with very high congestion, analyze the cause of the congestion. Issues such as high fan-out nets not using global resources, an improperly chosen optimization goal (speed versus area), very restrictive floorplan assignments, or the coding style can cause routing congestion. After you identify the cause, modify the source or settings to reduce routing congestion.

Related Links

- Fitter Resources Reports
  In Quartus Prime Help
- Analyzing and Optimizing the Design Floorplan on page 190
  This chapter discusses how the Chip Planner and LogicLock Plus regions help you improve your design's floorplan.

11.2 Optimizing Resource Utilization

The following lists the stages after design analysis:

1. Optimize resource utilization—Ensure that you have already set the basic constraints
2. I/O timing optimization—Optimize I/O timing after you optimize resource utilization and your design fits in the desired target device
3. Register-to-register timing optimization

Related Links

- Design Optimization Overview on page 102
- Timing Closure and Optimization on page 116

11.2.1 Resource Utilization Issues Overview

Resource utilization issues can be divided into three categories:

- Issues relating to *I/O pin utilization or placement*, including dedicated I/O blocks such as PLLs or LVDS transceivers.
- Issues relating to *logic utilization or placement*, including logic cells containing registers and LUTs as well as dedicated logic, such as memory blocks and DSP blocks.
- Issues relating to *routing*.

11.2.2 I/O Pin Utilization or Placement

Resolve I/O resource problems with these guidelines.

11.2.2.1 Guideline: Modify Pin Assignments or Choose a Larger Package

If a design that has pin assignments fails to fit, compile the design without the pin assignments to determine whether a fit is possible for the design in the specified device and package. You can use this approach if a Quartus Prime error message indicates fitting problems due to pin assignments.

If the design fits when all pin assignments are ignored or when several pin assignments are ignored or moved, you might have to modify the pin assignments for the design or select a larger package.

If the design fails to fit because insufficient I/Os pins are available, a successful fit can often be obtained by using a larger device package (which can be the same device density) that has more available user I/O pins.

Related Links

Managing Device I/O Pins on page 18

11.2.3 Logic Utilization or Placement

Resolve logic resource problems, including logic cells containing registers and LUTs, as well as dedicated logic such as memory blocks and DSP blocks, with these guidelines.

11.2.3.1 Guideline: Optimize Source Code

If your design does not fit because of logic utilization, then evaluate and modify the design at the source.You can often improve logic significantly by making design-specific changes to your source code. This is typically the most effective technique for improving the quality of your results.

If your design does not fit into available logic elements (LEs) or ALMs, but you have unused memory or DSP blocks, check if you have code blocks in your design that describe memory or DSP functions that are not being inferred and placed in dedicated logic. You might be able to modify your source code to allow these functions to be placed into dedicated memory or DSP resources in the target device.

Ensure that your state machines are recognized as state machine logic and optimized appropriately in your synthesis tool. State machines that are recognized are generally optimized better than if the synthesis tool treats them as generic logic. In the Quartus Prime software, you can check for the State Machine report under **Analysis & Synthesis** in the Compilation Report. This report provides details, including the state encoding for each state machine that was recognized during compilation. If your state machine is not being recognized, you might have to change your source code to enable it to be recognized.

Related Links

- AN 584: Timing Closure Methodology for Advanced FPGA Designs
- Recommended HDL Coding Styles
  In *Quartus Prime Pro Edition Handbook Volume 1*

11.2.3.2 Guideline: Optimize Synthesis for Area, Not Speed

If your design fails to fit because it uses too much logic, resynthesize the design to improve the area utilization. First, ensure that you have set your device and timing constraints correctly in your synthesis tool. Particularly when area utilization of the

design is a concern, ensure that you do not over-constrain the timing requirements for the design. Synthesis tools generally try to meet the specified requirements, which can result in higher device resource usage if the constraints are too aggressive.

If resource utilization is an important concern, you can optimize for area instead of speed.

- If you are using Quartus Prime synthesis, click **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Synthesis)** and select **Balanced** or **Area** for the **Optimization Technique**.
- If you want to reduce area for specific modules in your design using the **Area** or **Speed** setting while leaving the default **Optimization Technique** setting at **Balanced**, use the Assignment Editor.
- You can also use the **Speed Optimization Technique for Clock Domains** logic option to specify that all combinational logic in or between the specified clock domain(s) is optimized for speed.
- In some synthesis tools, not specifying an f_{MAX} requirement can result in less resource utilization.

Optimizing for area or speed can affect the register-to-register timing performance.

Note: In the Quartus Prime software, the **Balanced** setting typically produces utilization results that are very similar to those produced by the **Area** setting, with better performance results. The **Area** setting can give better results in some cases.

The Quartus Prime software provides additional attributes and options that can help improve the quality of your synthesis results.

Related Links

Optimization Mode
In Quartus Prime Help

11.2.3.3 Guideline: Restructure Multiplexers

Multiplexers form a large portion of the logic utilization in many FPGA designs. By optimizing your multiplexed logic, you can achieve a more efficient implementation in your Intel device.

Related Links

- Restructure Multiplexers logic option
  For more information about the Restructure Multiplexers option
- Recommended HDL Coding Styles
  For design guidelines to achieve optimal resource utilization for multiplexer designs

11.2.3.4 Guideline: Perform WYSIWYG Primitive Resynthesis with Balanced or Area Setting

The **Perform WYSIWYG Primitive Resynthesis** logic option specifies whether to perform WYSIWYG primitive resynthesis during synthesis. This option uses the setting specified in the **Optimization Technique** logic option. The **Perform WYSIWYG Primitive Resynthesis** logic option is useful for resynthesizing some or all of the

WYSIWYG primitives in your design for better area or performance. However, WYSIWYG primitive resynthesis can be done only when you use third-party synthesis tools.

Note: The **Balanced** setting typically produces utilization results that are very similar to the **Area** setting with better performance results. The **Area** setting can give better results in some cases. Performing WYSIWYG resynthesis for area in this way typically reduces register-to-register timing performance.

Related Links

Perform WYSIWYG Primitive Resynthesis logic option
For information about this logic option

11.2.3.5 Guideline: Use Register Packing

The **Auto Packed Registers** option implements the functions of two cells into one logic cell by combining the register of one cell in which only the register is used with the LUT of another cell in which only the LUT is used.

Related Links

Auto Packed Registers logic option
For more information about the Auto Packed Registers logic option

11.2.3.6 Guideline: Remove Fitter Constraints

A design with conflicting constraints or constraints that are difficult to meet may not fit in the targeted device. For example, a design might fail to fit if the location or LogicLock Plus assignments are too strict and not enough routing resources are available on the device.

To resolve routing congestion caused by restrictive location constraints or LogicLock Plus region assignments, use the **Routing Congestion** task in the Chip Planner to locate routing problems in the floorplan, then remove any internal location or LogicLock Plus region assignments in that area. If your design still does not fit, the design is over-constrained. To correct the problem, remove all location and LogicLock Plus assignments and run successive compilations, incrementally constraining the design before each compilation. You can delete specific location assignments in the Assignment Editor or the Chip Planner. To remove LogicLock Plus assignments in the Chip Planner, in the LogicLock Plus Regions Window, or on the Assignments menu, click **Remove Assignments.** Turn on the assignment categories you want to remove from the design in the **Available assignment categories** list.

Related Links

Analyzing and Optimizing the Design Floorplan on page 190
This chapter discusses how the Chip Planner and LogicLock Plus regions help you improve your design's floorplan.

11.2.3.7 Guideline: Flatten the Hierarchy During Synthesis

Synthesis tools typically provide the option of preserving hierarchical boundaries, which can be useful for verification or other purposes. However, the Quartus Prime software optimizes across hierarchical boundaries so as to perform the most logic minimization, which can reduce area in a design with no design partitions.

11.2.3.8 Guideline: Retarget Memory Blocks

If your design fails to fit because it runs out of device memory resources, your design may require a certain type of memory that the device does not have.

If the memory block was created with a parameter editor, open the parameter editor and edit the RAM block type so it targets a new memory block size.

ROM and RAM memory blocks can also be inferred from your HDL code, and your synthesis software can place large shift registers into memory blocks by inferring the Shift register (RAM-based) IP core. This inference can be turned off in your synthesis tool to cause the memory or shift registers to be placed in logic instead of in memory blocks. Also, for improved timing performance, you can turn this inference off to prevent registers from being moved into RAM.

Depending on your synthesis tool, you can also set the RAM block type for inferred memory blocks. In Quartus Prime synthesis, set the **ramstyle** attribute to the desired memory type for the inferred RAM blocks, or set the option to **logic**, to implement the memory block in standard logic instead of a memory block.

Consider the Resource Utilization by Entity report in the report file and determine whether there is an unusually high register count in any of the modules. Some coding styles can prevent the Quartus Prime software from inferring RAM blocks from the source code because of the blocks' architectural implementation, and force the software to implement the logic in flipflops. As an example, a function such as an asynchronous reset on a register bank might make the resistor bank incompatible with the RAM blocks in the device architecture, so that the register bank is implemented in flipflops. It is often possible to move a large register bank into RAM by slight modification of associated logic.

11.2.3.9 Guideline: Use Physical Synthesis Options to Reduce Area

The physical synthesis options available at **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Fitter)** help you decrease resource usage. When you enable physical synthesis, the Quartus Prime software makes placement-specific changes to the netlist that reduce resource utilization for a specific Intel device.

Note: Physical synthesis increases compilation time. To reduce the impact on compilation time, you can apply physical synthesis options to specific instances.

Related Links

Advanced Fitter Settings Dialog Box
In Quartus Prime Help

11.2.3.10 Guideline: Retarget or Balance DSP Blocks

A design might not fit because it requires too many DSP blocks. You can implement all DSP block functions with logic cells, so you can retarget some of the DSP blocks to logic to obtain a fit.

If the DSP function was created with the parameter editor, open the parameter editor and edit the function so it targets logic cells instead of DSP blocks. The Quartus Prime software uses the `DEDICATED_MULTIPLIER_CIRCUITRY` IP core parameter to control the implementation.

DSP blocks also can be inferred from your HDL code for multipliers, multiply-adders, and multiply-accumulators. You can turn off this inference in your synthesis tool. When you are using Quartus Prime synthesis, you can disable inference by turning off the **Auto DSP Block Replacement** logic option for your entire project. Click **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Synthesis)**. Turn off **Auto DSP Block Replacement**. Alternatively, you can disable the option for a specific block with the Assignment Editor.

The Quartus Prime software also offers the **DSP Block Balancing** logic option, which implements DSP block elements in logic cells or in different DSP block modes. The default **Auto** setting allows DSP block balancing to convert the DSP block slices automatically as appropriate to minimize the area and maximize the speed of the design. You can use other settings for a specific node or entity, or on a project-wide basis, to control how the Quartus Prime software converts DSP functions into logic cells and DSP blocks. Using any value other than **Auto** or **Off** overrides the `DEDICATED_MULTIPLIER_CIRCUITRY` parameter used in IP core variations.

11.2.3.11 Guideline: Use a Larger Device

If a successful fit cannot be achieved because of a shortage of routing resources, you might require a larger device.

11.2.4 Routing

Resolve routing resource problems with these guidelines.

11.2.4.1 Guideline: Set Auto Packed Registers to Sparse or Sparse Auto

The **Auto Packed Registers** option reduces LE or ALM count in a design.You can set this option by clicking **Assignment ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Fitter)**.

Related Links

Auto Packed Registers logic option

11.2.4.2 Guideline: Set Fitter Aggressive Routability Optimizations to Always

The **Fitter Aggressive Routability Optimization** option is useful if your design does not fit due to excessive routing wire utilization.

If there is a significant imbalance between placement and routing time (during the first fitting attempt), it might be because of high wire utilization. Turning on the **Fitter Aggressive Routability Optimizations** option can reduce your compilation time.

On average, this option can save up to 6% wire utilization, but can also reduce performance by up to 4%, depending on the device.

Related Links

Fitter Aggressive Routability Optimizations logic option

11.2.4.3 Guideline: Increase Router Effort Multiplier

The Router Effort Multiplier controls how quickly the router tries to find a valid solution. The default value is 1.0 and legal values must be greater than 0. Numbers higher than 1 help designs that are difficult to route by increasing the routing effort.

Numbers closer to 0 (for example, 0.1) can reduce router runtime, but usually reduce routing quality slightly. Experimental evidence shows that a multiplier of 3.0 reduces overall wire usage by approximately 2%. Using a Router Effort Multiplier higher than the default value could be beneficial for designs with complex datapaths with more than five levels of logic. However, congestion in a design is primarily due to placement, and increasing the Router Effort Multiplier does not necessarily reduce congestion.

Note: Any Router Effort Multiplier value greater than 4 only increases by 10% for every additional 1. For example, a value of 10 is actually 4.6.

11.2.4.4 Guideline: Remove Fitter Constraints

A design with conflicting constraints or constraints that are difficult to meet may not fit in the targeted device. For example, a design might fail to fit if the location or LogicLock Plus assignments are too strict and not enough routing resources are available on the device.

To resolve routing congestion caused by restrictive location constraints or LogicLock Plus region assignments, use the **Routing Congestion** task in the Chip Planner to locate routing problems in the floorplan, then remove any internal location or LogicLock Plus region assignments in that area. If your design still does not fit, the design is over-constrained. To correct the problem, remove all location and LogicLock Plus assignments and run successive compilations, incrementally constraining the design before each compilation. You can delete specific location assignments in the Assignment Editor or the Chip Planner. To remove LogicLock Plus assignments in the Chip Planner, in the LogicLock Plus Regions Window, or on the Assignments menu, click **Remove Assignments.** Turn on the assignment categories you want to remove from the design in the **Available assignment categories** list.

Related Links

Analyzing and Optimizing the Design Floorplan on page 190
This chapter discusses how the Chip Planner and LogicLock Plus regions help you improve your design's floorplan.

11.2.4.5 Guideline: Optimize Synthesis for Area, Not Speed

In some cases, resynthesizing the design to improve the area utilization can also improve the routability of the design. First, ensure that you have set your device and timing constraints correctly in your synthesis tool. Ensure that you do not overconstrain the timing requirements for the design, particularly when the area utilization of the design is a concern. Synthesis tools generally try to meet the specified requirements, which can result in higher device resource usage if the constraints are too aggressive.

If resource utilization is an important concern, you can optimize for area instead of speed.

- If you are using Quartus Prime synthesis, click **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Synthesis)** and select **Balanced** or **Area** for the **Optimization Technique**.
- If you want to reduce area for specific modules in your design using the **Area** or **Speed** setting while leaving the default **Optimization Technique** setting at **Balanced**, use the Assignment Editor.
- You can also use the **Speed Optimization Technique for Clock Domains** logic option to specify that all combinational logic in or between the specified clock domain(s) is optimized for speed.
- In some synthesis tools, not specifying an f_{MAX} requirement can result in less resource utilization.

Optimizing for area or speed can affect the register-to-register timing performance.

Note: In the Quartus Prime software, the **Balanced** setting typically produces utilization results that are very similar to those produced by the **Area** setting, with better performance results. The **Area** setting can give better results in some cases.

The Quartus Prime software provides additional attributes and options that can help improve the quality of your synthesis results.

Related Links

Optimization Mode
In Quartus Prime Help

11.2.4.6 Guideline: Optimize Source Code

If your design does not fit because of routing problems and the methods described in the preceding sections do not sufficiently improve the routability of the design, modify the design at the source to achieve the desired results. You can often improve results significantly by making design-specific changes to your source code, such as duplicating logic or changing the connections between blocks that require significant routing resources.

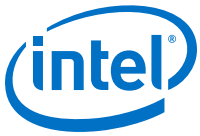
11.2.4.7 Guideline: Use a Larger Device

If a successful fit cannot be achieved because of a shortage of routing resources, you might require a larger device.

11.3 Scripting Support

You can run procedures and make settings described in this chapter in a Tcl script. You can also run some procedures at a command prompt. For detailed information about scripting command options, refer to the Quartus Prime command-line and Tcl API Help browser. To run the Help browser, type the following command at the command prompt:

```
quartus_sh --qhelp
```



You can specify many of the options described in this section either in an instance, or at a global level, or both.

Use the following Tcl command to make a global assignment:

```
set_global_assignment -name <.qsf variable name> <value>
```

Use the following Tcl command to make an instance assignment:

```
set_instance_assignment -name <.qsf variable name> <value> \
-to <instance name>
```

Note: If the <*value*> field includes spaces (for example, 'Standard Fit'), you must enclose the value in straight double quotation marks.

Related Links

- Tcl Scripting on page 72
- Quartus Prime Pro Edition Settings File Reference Manual
  For information about all settings and constraints in the Quartus Prime software.
- Command Line Scripting on page 65
  FPGA design software that easily integrates into your design flow saves time and improves productivity. The Intel Quartus Prime software provides you with a command-line executable for each step of the FPGA design flow to make the design process customizable and flexible.

11.3.1 Initial Compilation Settings

Use the Quartus Prime Settings File (`.qsf`) variable name in the Tcl assignment to make the setting along with the appropriate value. The **Type** column indicates whether the setting is supported as a global setting, an instance setting, or both.

Table 48. Advanced Compilation Settings

Setting Name	.qsf File Variable Name	Values	Type
Placement Effort Multiplier	PLACEMENT_EFFORT_MULTIPLIER	Any positive, non-zero value	Global
Router Effort Multiplier	ROUTER_EFFORT_MULTIPLIER	Any positive, non-zero value	Global
Router Timing Optimization level	ROUTER_TIMING_OPTIMIZATION_LEVEL	NORMAL, MINIMUM, MAXIMUM	Global
Final Placement Optimization	FINAL_PLACEMENT_OPTIMIZATION	ALWAYS, AUTOMATICALLY, NEVER	Global

11.3.2 Resource Utilization Optimization Techniques

This table lists the `.qsf` file variable name and applicable values for Resource Utilization Optimization settings.

Table 49. Resource Utilization Optimization Settings

Setting Name	.qsf File Variable Name	Values	Type
Auto Packed Registers [2]	AUTO_PACKED_REGISTERS_<device family name>	OFF, NORMAL, MINIMIZE AREA, MINIMIZE AREA WITH CHAINS, AUTO	Global, Instance
Perform WYSIWYG Primitive Resynthesis	ADV_NETLIST_OPT_SYNTH_WYSIWYG_REMAP	ON, OFF	Global, Instance
Optimization Technique	<device family name>_OPTIMIZATION_TECHNIQUE	AREA, SPEED, BALANCED	Global, Instance
Speed Optimization Technique for Clock Domains	SYNTH_CRITICAL_CLOCK	ON, OFF	Instance
State Machine Encoding	STATE_MACHINE_PROCESSING	AUTO, ONE-HOT, GRAY, JOHNSON, MINIMAL BITS, ONE-HOT, SEQUENTIAL, USER-ENCODE	Global, Instance
Auto RAM Replacement	AUTO_RAM_RECOGNITION	ON, OFF	Global, Instance
Auto ROM Replacement	AUTO_ROM_RECOGNITION	ON, OFF	Global, Instance
Auto Shift Register Replacement	AUTO_SHIFT_REGISTER_RECOGNITION	ON, OFF	Global, Instance
Auto Block Replacement	AUTO_DSP_RECOGNITION	ON, OFF	Global, Instance
Number of Processors for Parallel Compilation	NUM_PARALLEL_PROCESSORS	Integer between 1 and 16 inclusive, or ALL	Global

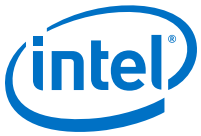
11.4 Document Revision History

Table 50. Document Revision History

Date	Version	Changes
2017.05.08	17.0.0	• Removed information about deprecated Integrated Synthesis • Revised topics: *Resolving Resource Utilization Issues*, *Guideline: Optimize Synthesis for Area, Not Speed*
2016.10.31	16.1.0	• Implemented Intel rebranding.
2016.05.02	16.0.0	• Removed information about deprecated physical synthesis options.
2015.11.02	15.1.0	Changed instances of *Quartus II* to *Quartus Prime*.

continued...

2 Allowed values for this setting depend on the device family that you select.



Date	Version	Changes
2014.12.15	14.1.0	Updated location of Fitter Settings, Analysis & Synthesis Settings, and Physical Synthesis Optimizations to Compiler Settings.
June 2014	14.0.0	• Removed Cyclone III and Stratix III devices references. • Removed Macrocell-Based CPLDs related information. • Updated template.
May 2013	13.0.0	Initial release.

12 Analyzing and Optimizing the Design Floorplan

As FPGA designs grow larger in density, the ability to analyze the design for performance, routing congestion, and logic placement is critical to meet the design requirements. This chapter discusses how the Chip Planner and LogicLock Plus regions help you improve your design's floorplan.

Design floorplan analysis helps to close timing, and ensure optimal performance in highly complex designs. With analysis capability, the Quartus Prime Chip Planner helps you close timing quickly on your designs. You can use the Chip Planner together with LogicLock Plus regions to compile your designs hierarchically and assist with floorplanning. Additionally, use partitions to preserve placement and routing results from individual compilation runs.

You can perform design analysis, as well as create and optimize the design floorplan with the Chip Planner. To make I/O assignments, use the Pin Planner.

Related Links

- Managing Device I/O Pins on page 18
- Intel FPGA Technical Training

12.1 Design Floorplan Analysis in the Chip Planner

The Chip Planner simplifies floorplan analysis by providing visual display of chip resources. With the Chip Planner, you can view post-compilation placement, connections, and routing paths. You can also make assignment changes, such as creating and deleting resource assignments.

The Chip Planner showcases:

- LogicLock Plus regions
- Relative resource usage
- Detailed routing information
- Fan-in and fan-out connections between nodes
- Timing paths between registers
- Delay estimates for paths
- Routing congestion information

12.1.1 Starting the Chip Planner

To start the Chip Planner, select **Tools ➤ Chip Planner**. You can also start the Chip Planner by the following methods:

- Click the Chip Planner icon on the Quartus Prime software toolbar.
- In the following tools, right-click any chip resource and select **Locate ➤ Locate in Chip Planner**:
  - Compilation Report
  - **LogicLock Plus Regions Window**
  - Technology Map Viewer
  - **Project Navigator** window
  - Node Finder
  - Simulation Report
  - Report Timing panel of the TimeQuest Timing Analyzer

12.1.2 Chip Planner GUI Components

12.1.2.1 Chip Planner Toolbar

The Chip Planner toolbar provides powerful tools for visual design analysis. You can access Chip Planner commands either from the **View** menu, or by clicking the icons in the toolbars.

12.1.2.2 Layers Settings

The Chip Planner allows you to control the display of resources.

Layers Settings Pane

With the **Layers Settings** pane, you can manage the graphic elements that the Chip Planner displays.

You open the **Layers Settings** pane by clicking **View ➤ Layers Settings**. The **Layers Settings** pane offers layer presets, which group resources that are often used together. The **Basic**, **Detailed**, and **Floorplan Editing** default presets are useful for general assignment-related activities. You can also create custom presets tailored to your needs.

Related Links

- Viewing Architecture-Specific Design Information on page 192
  The Chip Planner allows you to view architecture-specific information related to your design.
- Layers Settings Dialog Box
  In Quartus Prime Help

12.1.2.3 Locate History Window

As you optimize your design floorplan, you might have to locate a path or node in the Chip Planner many times. The **Locate History** window lists all the nodes and paths you have displayed using a **Locate in Chip Planner** command, providing easy access to the nodes and paths of interest to you.

If you locate a required path from the **TimeQuest Timing Analyzer Report Timing** pane, the **Locate History** window displays the required clock path. If you locate an arrival path from the **TimeQuest Timing Analyzer Report Timing** pane, the **Locate History** window displays the path from the arrival clock to the arrival data. Double-clicking a node or path in the **Locate History** window displays the selected node or path in the Chip Planner.

12.1.2.4 Chip Planner Floorplan Views

The Chip Planner uses a hierarchical zoom viewer that shows various abstraction levels of the targeted Intel device. As you zoom in, the level of abstraction decreases, revealing more details about your design.

Bird's Eye View

The Bird's Eye View displays a high-level picture of resource usage for the entire chip and provides a fast and efficient way to navigate between areas of interest in the Chip Planner.

The Bird's Eye View is particularly useful when the parts of your design that you want to view are at opposite ends of the chip, and you want to quickly navigate between resource elements without losing your frame of reference.

Properties Window

The **Properties** window displays detailed properties of the objects (such as atoms, paths, LogicLock Plus regions, or routing elements) currently selected in the Chip Planner. To display the **Properties** window, right-click the object and select **View ➤ Properties**.

Related Links

Bird's Eye View Window
In Quartus Prime Help

12.1.3 Viewing Architecture-Specific Design Information

The Chip Planner allows you to view architecture-specific information related to your design. By enabling the options in the **Layers Settings** pane, you can view:

- **Device routing resources used by your design**—View how blocks are connected, as well as the signal routing that connects the blocks.
- **LE configuration**—View logic element (LE) configuration in your design. For example, you can view which LE inputs are used; whether the LE utilizes the register, the look-up table (LUT), or both; as well as the signal flow through the LE.
- **ALM configuration**—View ALM configuration in your design. For example, you can view which ALM inputs are used; whether the ALM utilizes the registers, the upper LUT, the lower LUT, or all of them. You can also view the signal flow through the ALM.
- **I/O configuration**—View device I/O resource usage. For example, you can view which components of the I/O resources are used, whether the delay chain settings are enabled, which I/O standards are set, and the signal flow through the I/O.
- **PLL configuration**—View phase-locked loop (PLL) configuration in your design. For example, you can view which control signals of the PLL are used with the settings for your PLL.
- **Timing**—View the delay between the inputs and outputs of FPGA elements. For example, you can analyze the timing of the DATAB input to the COMBOUT output.

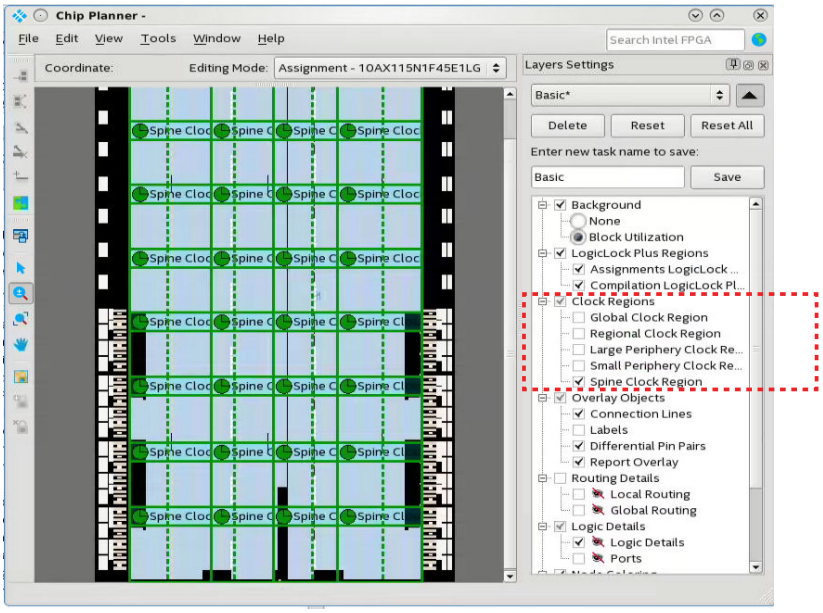
Related Links

- Layers Settings on page 191
  The Chip Planner allows you to control the display of resources.
- Layers Settings Dialog Box
  In Quartus Prime Help

12.1.4 Viewing Available Clock Networks in the Device

When you enable a clock region layer in the **Layers Settings** pane, you display the areas of the chip that are driven by global and regional clock networks. When the selected device does not contain a given clock region, the option for that category is unavailable in the dialog box.

Figure 72. Clock Regions



- Depending on the clock layers that you activate in the **Layers Settings** pane, the Chip Planner displays regional and global clock regions in the device, and the connectivity between clock regions, pins, and PLLs.
- Clock regions appear as rectangular overlay boxes with labels indicating the clock type and index. Select a clock network region by clicking the clock region. The clock-shaped icon at the top-left corner indicates that the region represents a clock network region.
- Spine/sector clock regions have a dotted vertical line in the middle. This dotted line indicates where two columns of row clocks meet in a sector clock.
- To change the color in which the Chip Planner displays clock regions, select **Tools ➤ Options ➤ Colors ➤ Clock Regions**.

Related Links

- Spine Clock Limitations on page 130
  If your project has high clock routing demands, due to limitations in the Quartus Prime software, you may see spine clock errors.
- Layers Settings on page 191
  The Chip Planner allows you to control the display of resources.
- Report Spine Clock Utilization dialog box (Chip Planner)
  In Quartus Prime Help

12.1.5 Viewing Routing Congestion

The **Report Routing Utilization** task allows you to determine the percentage of routing resources in use following a compilation. This feature can identify zones with lack of routing resources, helping you to make design changes to meet routing congestion design requirements.

To view the routing congestion in the Chip Planner:

1. In the **Tasks** pane, double-click the **Report Routing Utilization** command to launch the **Report Routing Utilization** dialog box.
2. Click **Preview** in the **Report Routing Utilization** dialog box to preview the default congestion display.
3. Change the **Routing Utilization Type** to display congestion for specific resources.
   The default display uses dark blue for 0% congestion (blue indicates zero utilization) and red for 100%. You can adjust the slider for **Threshold percentage** to change the congestion threshold level.

The congestion map helps you determine whether you can modify the floorplan, or modify the RTL to reduce routing congestion. Consider:

- The routing congestion map uses the color and shading of logic resources to indicate relative resource utilization; darker shading represents a greater utilization of routing resources. Areas where routing utilization exceeds the threshold value specified in the **Report Routing Utilization** dialog box appear in red.
- To identify a lack of routing resources, it is necessary to investigate each routing interconnect type separately by selecting each interconnect type in turn in the **Routing Utilization Settings** dialog box.
- The Compiler's messages contain information about average and peak interconnect usage. Peak interconnect usage over 75%, or average interconnect usage over 60%, could be an indication that it might be difficult to fit your design. Similarly, peak interconnect usage over 90%, or average interconnect usage over 75%, are likely to have increased chances of not getting a valid fit.

Related Links

Viewing Routing Resources on page 198
With the Chip Planner and the **Locate History** window, you can view the routing resources that a path or connection uses.

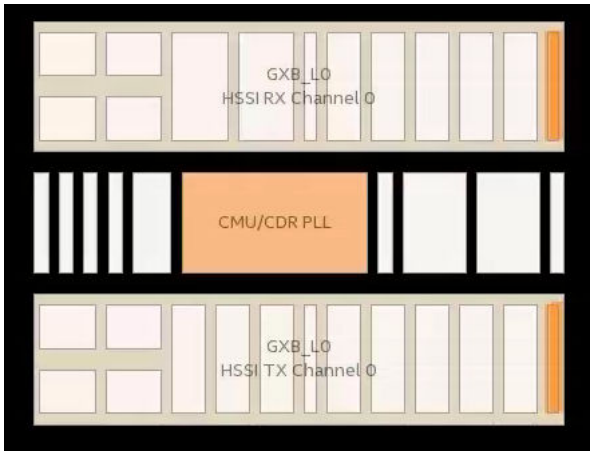
12.1.6 Viewing I/O Banks

To view the I/O bank map of the device in the Chip Planner, double-click **Report All I/O Banks** in the **Tasks** pane.

12.1.7 Viewing High-Speed Serial Interfaces (HSSI)

The Chip Planner displays a detailed block view of the receiver and transmitter channels of the high-speed serial interfaces. To display the HSSI block view, double-click **Report HSSI Block Connectivity** in the **Tasks** pane.

Figure 73. Arria 10 HSSI Channel Blocks



12.1.8 Generating Fan-In and Fan-Out Connections

Displays the atoms that fan-in to or fan-out from a resource.

- To display the fan-in or fan-out connections from a resource you selected, use the **Generate Fan-In Connections** icon or the **Generate Fan-Out Connections** icon in the Chip Planner toolbar.
- To remove the connections displayed, use the **Clear Unselected Connections** icon in the Chip Planner toolbar. Alternatively, use the **View** menu.

12.1.9 Generating Immediate Fan-In and Fan-Out Connections

Displays the immediate fan-in or fan-out connection for the selected atom.

For example, when you view the immediate fan-in for a logic resource, you see the routing resource that drives the logic resource. You can generate immediate fan-ins and fan-outs for all logic resources and routing resources.

- To display the immediate fan-in or fan-out connections, click **View ➤ Generate Immediate Fan-In Connections** or **View ➤ Generate Immediate Fan-Out Connections**.
- To remove the connections displayed, use the **Clear Unselected Connections** icon in the Chip Planner toolbar.

12.1.10 Exploring Paths in the Chip Planner

Use the Chip Planner to explore paths between logic elements. The following examples use the Chip Planner to traverse paths from the Timing Analysis report.

12.1.10.1 Analyzing Connections for a Path

To determine the elements forming a selected path or connection in the Chip Planner, click the **Expand Connections** icon in the Chip Planner toolbar.

Related Links

Expand Connections Command (View Menu)
In Quartus Prime Help

12.1.10.2 Locate Path from the Timing Analysis Report to the Chip Planner

To locate a path from the Timing Analysis report to the Chip Planner, perform the following steps:

1. Select the path you want to locate in the Timing Analysis report.
2. Right-click the path and point to **Locate Path ➤ Locate in Chip Planner**. The path appears in the **Locate History** window of the Chip Planer.

Figure 74. Path List in the Locate History Window

Related Links

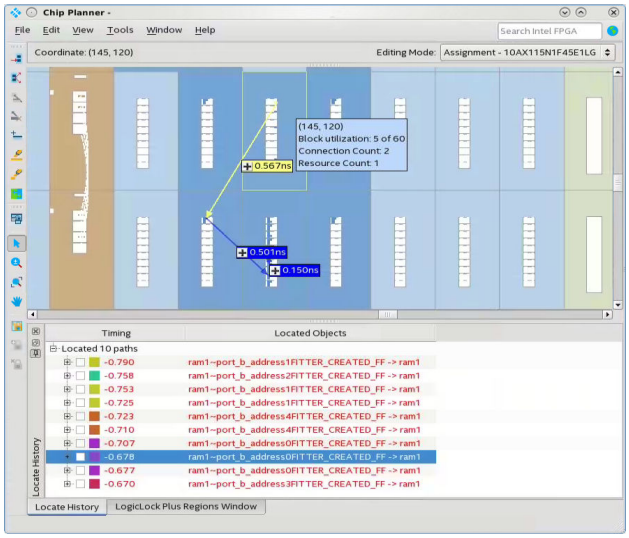
Displaying Path Reports with the TimeQuest Timing Analyzer on page 120
The TimeQuest timing analyzer generate reports with information about all valid register-to-register paths.

12.1.10.3 Show Delays

With the **Show Delays** feature, you can view timing delays for paths appearing in TimeQuest Timing Analyzer reports. To access this feature, click **View ➤ Show Delays** in the main menu. Alternatively click the Show Delays icon in the Chip Planner toolbar. To see the partial delays on the selected path, click the "+" sign next to the path delay displayed in the **Locate History** window.

For example, you can view the delay between two logic resources or between a logic resource and a routing resource.

Figure 75. Show Delays Associated in a TimeQuest Timing Analyzer Path



12.1.10.4 Viewing Routing Resources

With the Chip Planner and the **Locate History** window, you can view the routing resources that a path or connection uses. You can also select and display the Arrival Data path and the Arrival Clock path.

Figure 76. Show Physical Routing

In the **Locate History** window, right-click a path and select **Show Physical Routing** to display the physical path. To adjust the display, right-click and select **Zoom to Selection**.

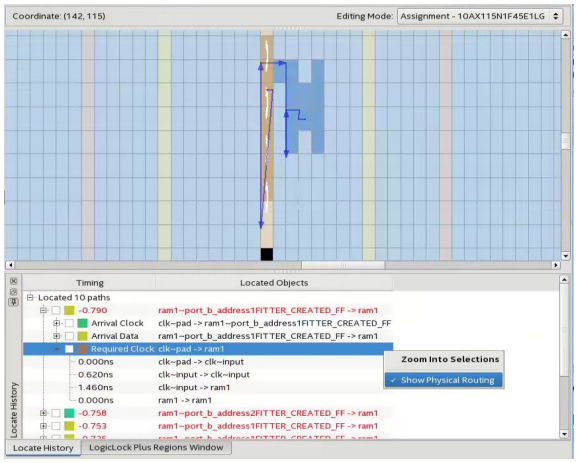
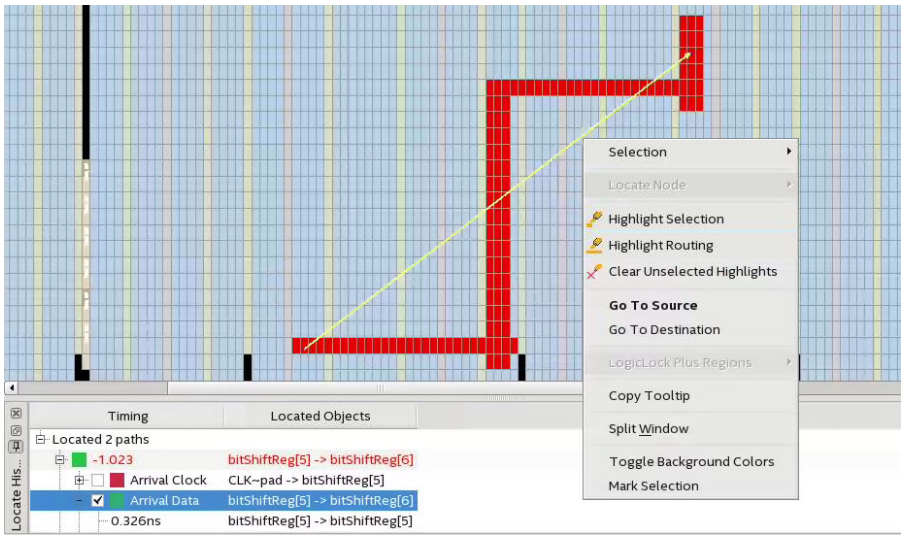


Figure 77. Highlight Routing

To see the rows and columns where the Fitter routed the path, right-click a path and select **Highlight Routing**.



Related Links

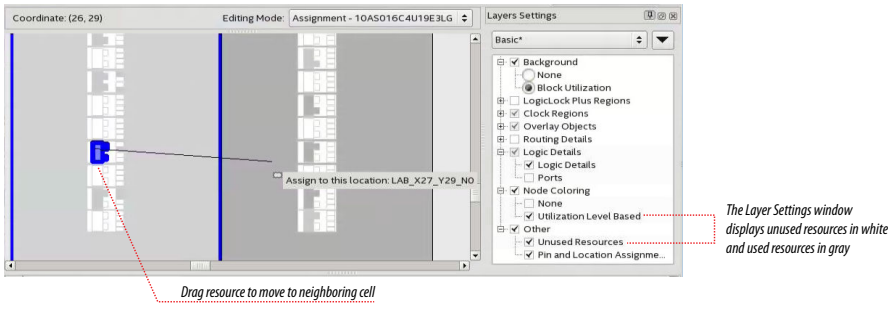
Viewing Routing Congestion on page 194
The **Report Routing Utilization** task allows you to determine the percentage of routing resources in use following a compilation.

12.1.11 Viewing Assignments in the Chip Planner

You can view location assignments in the Chip Planner by selecting the appropriate layer, or any custom preset that displays block utilization in the **Layers Settings** pane.

The Chip Planner displays assigned resources in a predefined color (gray, by default).

Figure 78. Viewing Assignments in the Chip Planner



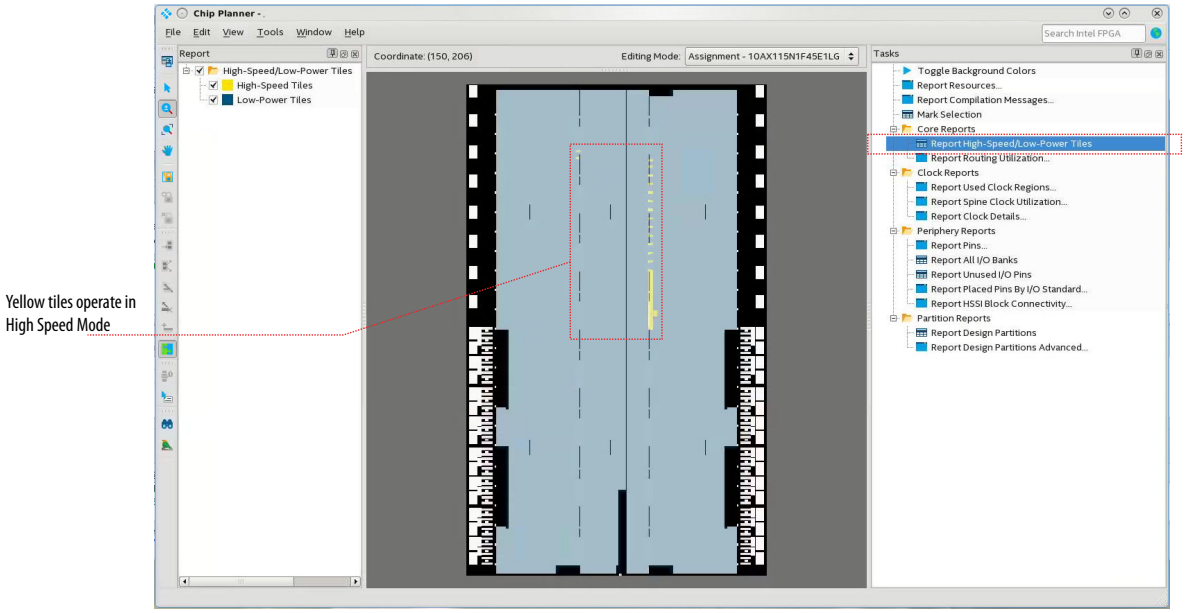
To create or move an assignment, or to make node and pin location assignments to LogicLock Plus regions, drag the selected resource to a new location. The Fitter applies the assignments that you create during the next place-and-route operation.

12.1.12 Viewing High-Speed and Low-Power Tiles in the Chip Planner

Some Intel devices have ALMs that can operate in either high-speed mode or low-power mode. The power mode is set during the fitting process in the Quartus Prime software. These ALMs are grouped together to form larger blocks, called "tiles".

To view a power map, double-click **Tasks ➤ Core Reports ➤ Report High-Speed/ Low-Power Tiles** after running the Fitter. The Chip Planner displays low-power and high-speed tiles in contrasting colors; yellow tiles operate in a high-speed mode, while blue tiles operate in a low-power mode.

Figure 79. High-Speed and Low Power Tiles in an Arria 10 Device



12.2 LogicLock Plus Regions

LogicLock Plus regions are floorplan location constraints. When you assign instances or nodes to a LogicLock Plus region, you direct the Fitter to place those instances or nodes within the region. A floorplan can contain multiple LogicLock Plus regions. Being constraints, LogicLock Plus Regions can have a negative effect on timing closure.

LogicLock Plus regions don't have preservation attributes, just boundaries and reservation of logic resources. You can use Quartus Prime Pro Edition software to implement fully hierarchical LogicLock Plus region assignments.

A LogicLock Plus region is composed of two elements:

- Placement Region: Constrains logic to a specific area of the device; the Fitter places the logic in the region you specify. If you have designated the region as reserved, the Fitter cannot place other logic in the region.
- Routing Region: Constrains routing to a specific area. The routing region must encompass the placement region. Routing regions cannot be set as Reserved. For more details, refer to *Defining Routing Regions*.

The attributes of a LogicLock Plus regions are:

Table 51. Attributes of LogicLock Plus Regions

Name	Value	Behavior
Width	Number of columns	Specifies the width of the LogicLock Plus region.
Height	Number of rows	Specifies the height of the LogicLock Plus region.
Origin	Any Floorplan Location	Specifies the location of the LogicLock Plus region on the floorplan. The origin is at the lower left corner of the LogicLock Plus region.
Reserved	Off | On	Prevents the Fitter from placing other logic in the region. You cannot apply the **Reserved** assignment to routing regions.
Core-Only	Off | On	Excludes periphery resources from a region. Unlike the Quartus Prime Standard Edition software, Quartus Prime Pro Edition region assignments apply to periphery resources by default. If the region is designated as **Reserved** and **Core Only**, periphery resources are not reserved from the region.
Routing Region	Unconstrained | Whole Chip | Fixed with Expansion | Custom	Type of routing region. For more details, refer to *Defining Routing Regions*.

Related Links

- Defining Routing Regions on page 205
  A routing region is an element of a LogicLock Plus region that specifies the routing area.
- Replace LogicLock Regions
  In *Quartus Prime Pro Edition Handbook Volume 1*

12.2.1 Migrating Assignments between Quartus Prime Standard Edition and Quartus Prime Pro Edition

The Quartus Prime Pro Edition software does not support the Quartus Prime Standard Edition LogicLock assignments. Therefore, if you are migrating a design from Quartus Prime Standard Edition to Quartus Prime Pro Edition, you must convert the LogicLock assignments into LogicLock Plus assignments.

Related Links

Migrating to Quartus Prime Pro Edition
In *Quartus Prime Pro Edition Handbook Volume 1*

12.2.2 LogicLock Plus Regions Window

The **LogicLock Plus Regions Window** provides a summary of all LogicLock Plus regions defined in your design. Use the **LogicLock Plus Regions Window** to create, assign elements, and modify properties of a LogicLock Plus region.

Open the **LogicLock Plus Regions Window** in the Chip Planner by clicking **View ➤ LogicLock Plus Window**, and in Quartus Prime by clicking **Assignments ➤ LogicLock Plus Window**.

Figure 80. LogicLock Plus Regions Window

Region Name	Width	Height	Origin	Reserved	Core-Only	Routing Region
LogicLock Plus Regions						
l[1].mod_1	20	7	X52_Y27	On	Off	Fixed with expansion 5
l[2].mod_1	Custom Shape	Custom Shape	Custom Shape	On	On	Whole chip
l[3].mod_1	20	20	X17_Y69	Off	Off	Unconstrained
l[4].mod_1	20	20	X28_Y26	On	On	Whole chip
l[5].mod_1	Custom Shape	Custom Shape	Custom Shape	Off	On	Unconstrained
<<new>>						

You can customize the **LogicLock Plus Regions Window** by dragging and dropping the columns to change their order; you can also show and hide optional columns by right-clicking any column heading and then selecting the appropriate columns in the shortcut menu.

LogicLock Plus Region Properties Dialog Box

Use the **LogicLock Plus Region Properties** dialog box to view and modify detailed information about your LogicLock Plus region, such as which entities and nodes are assigned to your region, and which resources are required.

To open the **LogicLock Plus Region Properties** dialog box, double-click any region in the **LogicLock Plus Regions Window**. Alternatively, right-click the region and select **LogicLock Plus Region Properties**.

12.2.3 Creating LogicLock Plus Regions

You can create LogicLock Plus Regions using several tools, such as the **Project Navigator**, **LogicLock Plus Regions Window**, or the **Chip Planner**.

Creating LogicLock Plus Regions with the Project Navigator

After you perform a full compilation or analysis and elaboration on the design, the Project Navigator displays the hierarchy of the design. If the Project Navigator is not already open, click **View ➤ Project Navigator**.

With the design hierarchy expanded, right-click any design entity, and select **Create New LogicLock Plus Region** to create a LogicLock Plus region, and assign the entity to the new region. The new region has the same name as the full hierarchy path of one of the nodes.

Creating LogicLock Plus Regions with the LogicLock Plus Regions Window

If the **LogicLock Plus Regions Window** is not open, click **Assignments ➤ LogicLock Plus Regions Window**. In the **LogicLock Plus Regions Window**, double-click **<<new>>**, and click **...** to open the Node Finder. Select nodes, and click **OK**. The new region has the name of the full hierarchy path of one of the nodes. If you don't select a node, the LogicLock Plus region creation cancels.

Creating LogicLock Plus Regions with the Chip Planner

To create a LogicLock Plus region in the Chip Planner, click the **Create New and Add LogicLock Plus Region** icon in the **Navigation** toolbar. On the Chip Planner floorplan, click and drag to create a region of your preferred location and size.

- You can also change the region shape and assign entities to the region. The order that you assign the entities or define the shape doesn't matter.
- Before compilation, ensure that your LogicLock Plus region has valid assignments.

Related Links

- Placing Device Resources into LogicLock Plus Regions on page 206
- Node Finder Command
  In Quartus Prime Help
- Creating LogicLock Plus Assignments with Tcl commands on page 209
  The Quartus Prime software supports Tcl commands to create or modify LogicLock Plus assignments.

12.2.3.1 Customizing the Shape of LogicLock Plus Regions

To create custom shaped LogicLock Plus regions, you can perform logic operations. When you create a floorplan for your design, you may want to create non-rectangular LogicLock Plus regions to exclude certain resources, or to place parts of your design around specific device resources to improve performance.

Attention: There is no undo feature for the LogicLock Plus shapes for 17.0. Take care when creating or modifying shapes.

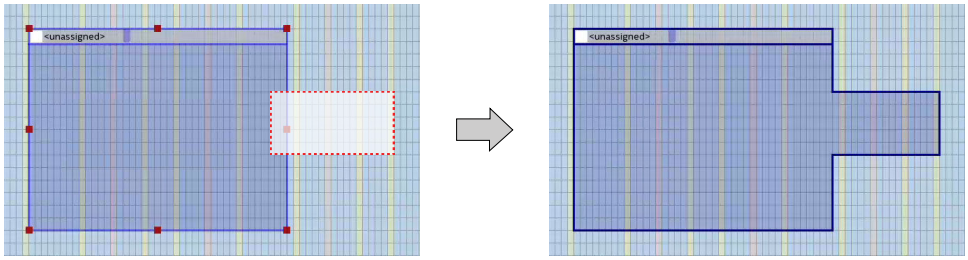
12.2.3.1.1 Adding a New Shape to a LogicLock Plus Region

To add a new shape to an existing LogicLock Plus region, perform the following steps in the Chip Planner:

1. Select the LogicLock Plus region.
2. In the **Navigation** toolbar, click the **Add LogicLock Plus Region** icon .
3. Click and drag to generate the shape you want to add. The new shape merges automatically with the selected LogicLock Plus region.

   Attention: If you selected more than one region, the operation appends the new shape to all of them.

Figure 81. Using the Add LogicLock Plus Region Feature



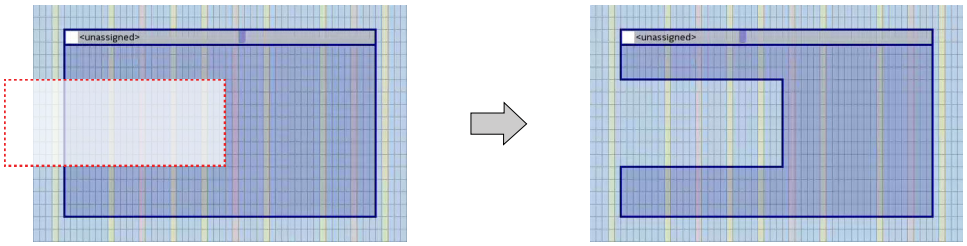
12.2.3.1.2 Subtracting Shape from LogicLock Plus Region

To subtract a shape from an existing LogicLock Plus region, perform the following steps in the Chip Planner:

1. Select the LogicLock Plus region.
2. In the **Navigation** toolbar, click the **Subtract LogicLock Plus Region** icon .
3. Click and drag the shape you want to subtract. The modified region displays automatically.

   The operation performs in all selected regions.

Figure 82. Using the Subtract LogicLock Plus Region Feature

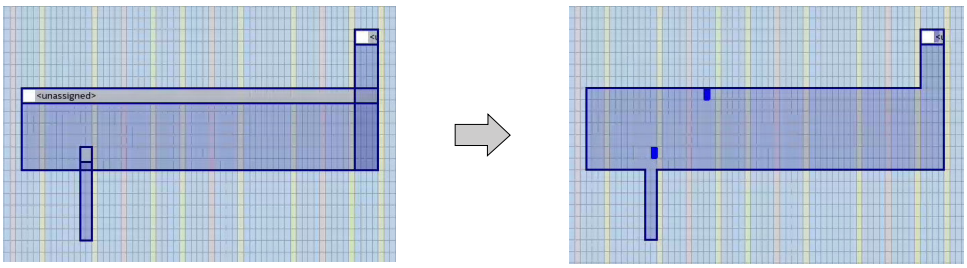


12.2.3.1.3 Merging LogicLock Plus Regions

To merge two or more LogicLock Plus regions, perform the following steps:

1. Ensure that no more than one of the regions that you intend to merge has logic assignments.
2. Arrange the regions into the locations where you want the resultant region.
3. Select all the individual regions that you want to merge by clicking each of them while pressing the Shift key.
4. Right-click the title bar of any of the selected LogicLock Plus regions and select **LogicLock Plus Regions ➤ Merge LogicLock Plus Region**. The individual regions that you select merge to create a single new region.
   If you select multiple named regions, the **Merge LogicLock Plus Region** option is deactivated.

Figure 83. Using the Merge LogicLock Plus Region command



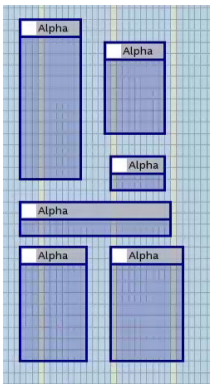
Related Links

Creating LogicLock Plus Regions on page 202
You can create LogicLock Plus Regions using several tools, such as the **Project Navigator**, **LogicLock Plus Regions Window**, or the **Chip Planner**.

12.2.3.2 Noncontiguous LogicLock Plus Regions

You can create disjointed regions by using the LogicLock Plus region manipulation tools. Noncontiguous regions act as a single LogicLock Plus region for all LogicLock Plus region attributes.

Figure 84. Noncontiguous LogicLock Plus Region



Related Links

Merging LogicLock Plus Regions on page 204

12.2.4 Defining Routing Regions

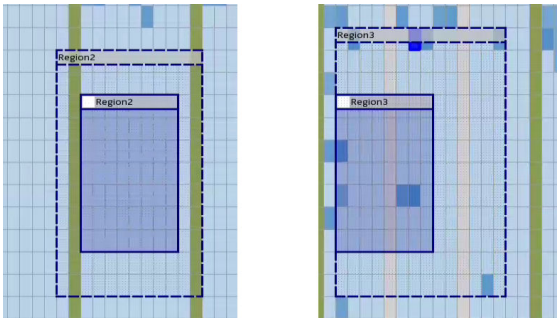
A routing region is an element of a LogicLock Plus region that specifies the routing area. A routing region must encompass the existing LogicLock Plus placement region. Routing regions cannot be set as reserved. To define the routing region, double-click the **Routing Region** cell in the **LogicLock Plus Regions** window, and select an option from the drop-down menu.

Valid routing region options are:

Table 52. Routing Region Options

Option	Description
Unconstrained (default)	Allows the fitter to use any available routes on the device.
Whole Chip	Same as Unconstrained, but writes the constraint in the Quartus Prime settings file (`.qsf`).
Fixed with Expansion	Follows the outline of the placement region. The routing region scales by a number of rows/cols larger than the placement region.
Custom	Allows you to make a custom shape routing region around the LogicLock Plus region. When you select the **Custom** option, the placement and routing regions move independently in the Chip Planner. In this case, move the placement and routing regions by selecting both using the Shift key.

Figure 85. Routing Regions



12.2.5 Placing Device Resources into LogicLock Plus Regions

You can assign an entity in the design to only one LogicLock Plus region, but the entity can inherit regions by hierarchy. This hierarchy allows a reserved region to have a subregion without reserving the resources in the subregion.

If a LogicLock Plus region boundary includes part of a device resource, the Quartus Prime software allocates the entire resource to that LogicLock Plus region.

To add an instance using the **LogicLock Plus Region** window, right-click the region and select **LogicLock Plus Properties ➤ Add**. Alternatively, in the Quartus Prime software you can drag and drop entities from the Hierarchy viewer into a LogicLock Plus region's name field in the **LogicLock Plus Regions Window.**

12.2.5.1 Pin Assignment

A LogicLock Plus region incorporates all device resources within its boundaries, including memory and pins. The Quartus Prime Pro Edition software does not include pins automatically when you assign an entity to a region, unless the **Core Only** attribute is off.

You can manually assign pins to LogicLock Plus regions; however, this placement puts location constraints on the region. The software only obeys pin assignments to locked regions that border the periphery of the device. The locked regions must include the I/O pins as resources.

12.2.5.2 Reserved LogicLock Plus Regions

The **Reserved** attribute instructs the Fitter to only place the entities and nodes that you specifically assigned to the LogicLock Plus region in the LogicLock Plus region.

The Quartus Prime software honors all entity and node assignments to LogicLock Plus regions. Occasionally entities and nodes do not occupy an entire region, which leaves some of the region’s resources unoccupied.

To increase the region’s resource utilization and performance, Quartus Prime software by default fills the unoccupied resources with other nodes and entities that have not been assigned to another region. To prevent this behavior, turn on **Reserved** in the **LogicLock Plus Regions** window.

12.2.5.3 Virtual Pins

A virtual pin is an I/O element that the Compiler temporarily maps to a logic element, and not to a pin during compilation. The software implements virtual pins as LUTs. To assign a Virtual Pin, use the Assignment Editor. You can create virtual pins by assigning the **Virtual Pin** logic option to an I/O element.

When you apply the **Virtual Pin** assignment to an input pin, the pin no longer appears as an FPGA pin; the Compiler fixes the virtual pin to GND in the design. The virtual pin is not a floating node.

Use virtual pins only for I/O elements in lower-level design entities that become nodes after you import the entity to the top-level design.

You might use virtual pin assignments when you compile a partial design, because not all the I/Os from a partial design drive chip pins at the top level.

The virtual pin assignment identifies the I/O ports of a design module that are internal nodes in the top-level design. These assignments prevent the number of I/O ports in the lower-level modules from exceeding the total number of available device pins.

Note: The **Virtual Pin** logic option must be assigned to an input or output pin. If you assign this option to a bidirectional pin, tri-state pin, or registered I/O element, Synthesis ignores the assignment. If you assign this option to a tri-state pin, the Fitter inserts an I/O buffer to account for the tri-state logic; therefore, the pin cannot be a virtual pin. You can use multiplexer logic instead of a tri-state pin if you want to continue to use the assigned pin as a virtual pin. Do not use tri-state logic except for signals that connect directly to device I/O pins.

In the top-level design, you connect these virtual pins to an internal node of another module. By making assignments to virtual pins, you can place those pins in the same location or region on the device as that of the corresponding internal nodes in the top-level module. You can use the **Virtual Pin** option when compiling a LogicLock Plus module with more pins than the target device allows. The **Virtual Pin** option can enable timing analysis of a design module that more closely matches the performance of the module after you integrate it into the top-level design.

To display all assigned virtual pins in the design with the Node Finder, you can set **Filter Type** to **Pins: Virtual**. To access the Node Finder from the Assignment Editor, double-click the **To** field; when the arrow appears on the right side of the field, click and select **Node Finder**.

Related Links

- Assigning Virtual Pins with a Tcl command on page 211
- Managing Device I/O Pins on page 18
- Node Finder Command (View Menu)
  In Quartus Prime Help

12.2.5.4 Placement Best Practices for Arria 10 FPGAs

For best results, consider the topology of Arria 10 FPGAs in your design.

Figure 86. I/O Columns in Arria 10 FPGAs

Arria 10 FPGAs have I/O columns located in the middle of the device. Signals can only enter or exit these columns from the side that faces the device edge.

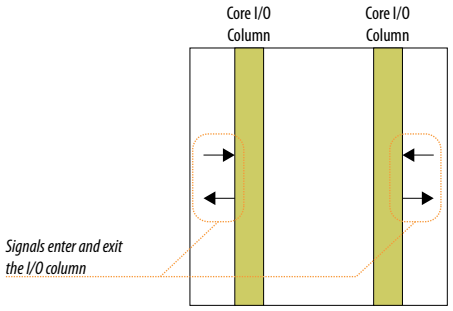


Figure 87. Signals Crossing I/O Columns in Arria 10 FPGAs

Routing a signal to cross the I/O column increases the routing delay, and can reduce design performance.

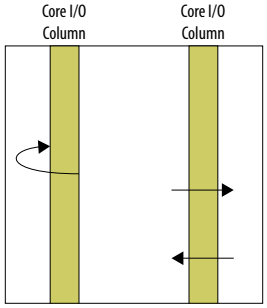
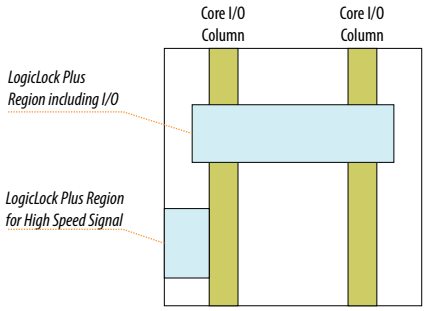


Figure 88. Strategic Placement for LogicLock Plus Regions in Arria 10 FPGAs

- If a LogicLock Plus region contains a register that interface with the I/O column, place the LogicLock Plus region so that the region covers the I/O column and the core logic, for better access to the I/O column adjacent to the outer column edge.
- For high speed signal, you can get best results if you place the LogicLock Plus region on the outside of the I/O column, because the fitter is less likely to cross the column and incur delay.



12.2.6 Hierarchical Regions

LogicLock Plus regions are fully hierarchical. Parent regions must completely contain all child regions. The **Reserved** and **Core-Only** assignments also apply hierarchically.

LogicLock Plus assignments follow the same precedence as other constraints and assignments.

You can assign an entity in the design to only one LogicLock Plus region, but the entity can inherit regions by hierarchy. This hierarchy allows a reserved region to have a subregion without reserving the resources in the subregion.

12.2.7 Additional Quartus Prime LogicLock Plus Design Features

To complement the **LogicLock Plus Regions Window**, the Quartus Prime software has additional features to help you design with LogicLock Plus regions.

12.2.7.1 Quartus Prime Revisions Feature

When you evaluate different LogicLock Plus regions in your design, you might want to experiment with different configurations to achieve your desired results. The Quartus Prime Revisions feature allows you to organize the same project with different settings until you find an optimum configuration.

To use the Revisions feature, choose **Project ➤ Revisions**. You can create a revision from the current design or any previously created revisions. Each revision can have an associated description. You can use revisions to organize the placement constraints created for your LogicLock Plus regions.

12.3 Scripting Support

You can run procedures and specify the settings described in this chapter in a Tcl script. You can also run some procedures at a command prompt.

Related Links

- Tcl Scripting on page 72
- API Functions for Tcl
  In Quartus Prime Help
- Quartus Prime Pro Edition Settings File Reference Manual
  For information about all settings and constraints in the Quartus Prime software.

12.3.1 Creating LogicLock Plus Assignments with Tcl commands

The Quartus Prime software supports Tcl commands to create or modify LogicLock Plus assignments.

Note: Specify node names by using the full hierarchy path to the node.

Create or Modify a Placement Region

You can create the LogicLock Plus region from the GUI, or add the region directly to the QSF. The QSF entry contains the X/Y coordinates of the vertices and the Placement Region name.

The following assignment creates a new placement region with bounding box coordinates X46 Y36 X65 Y49:

```
set_instance_assignment -name PLACE_REGION "X46 Y36 X65 Y49" -to <node name(s)>
```

- You can use the same command format to modify an existing assignment.
- To specify a non-rectangular or disjoint region, use a semicolon (;) as the delimiter between two or more bounding boxes.
- Assign multiple instances to the same region with multiple PLACE_REGION instance assignments.

Create or Modify a Routing Region

The following assignment creates a routing region with bounding box coordinates X5 Y5 X30 Y30:

```
set_instance_assignment -name ROUTE_REGION -to <node name(s)> "X5 Y5 X30 Y30"
```

- You can use the same command format to modify an existing assignment.
- All instances with a routing region assignment must have a respective placement region; the routing region must fully contain the placement region.

Specify a Region as Reserved

The following assignment reserves an existing region:

```
set_instance_assignment -name <instance name> RESERVE_PLACE_REGION -to <node
name(s)> ON
```

- You can only reserve placement regions.

Specify a Region as Core Only

By default, the Quartus Prime Pro Edition software includes pins in LogicLock Plus assignments. To specify a region as core only (i.e., periphery logic in the instance is not constrained), use the following assignment:

```
set_instance_assignment -name <instance name> CORE_ONLY_PLACE_REGION -to <node
name(s)> ON
```

Related Links

Creating LogicLock Plus Regions on page 202
You can create LogicLock Plus Regions using several tools, such as the **Project Navigator**, **LogicLock Plus Regions Window**, or the **Chip Planner**.

12.3.2 Assigning Virtual Pins with a Tcl command

Use the following Tcl command to turn on the virtual pin setting for a pin called `my_pin`:

```
set_instance_assignment -name VIRTUAL_PIN ON -to my_pin
```

Related Links

- Virtual Pins on page 207
  A virtual pin is an I/O element that the Compiler temporarily maps to a logic element, and not to a pin during compilation. The software implements virtual pins as LUTs.
- Managing Device I/O Pins on page 18
- Node Finder Command (View Menu)
  In Quartus Prime Help

12.3.3 LogicLock Plus Region Assignment Examples

These examples show the syntax for various LogicLock Plus region assignments in the `.qsf` file. Optionally enter these assignments in the Assignment Editor, the LogicLock Plus Regions Window, or the Chip Planner.

Example 6. Assign Rectangular LogicLock Plus Region

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

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

Example 7. Assign Non-Rectangular LogicLock Plus Region

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

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

Example 8. Assign Subordinate LogicLock Plus Instances

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

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

Example 9. Assign Multiple LogicLock Plus Instances

By default, a LogicLock Plus region constraint allows logic from other instances to share the same region. In other words, the following assignments are not in conflict. This assignment places instance c and instance g together. This may be useful if instance c and instance g are heavily interacting.

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

Example 10. Assigned Reserved LogicLock Plus Regions

Optionally reserve an entire LogicLock Plus region for one instance and any of its subordinate instances.

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

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

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

12.4 Document Revision History

Table 53. Document Revision History

Date	Version	Changes
2017.05.08	17.0.0	• Chapter reorganization and content update. • Added figures: Clock Regions, Path List in the Locate History Window, Show Physical Routing, Using the Add Rectangle Feature, Using the Subtract Rectangle Feature, Creating a Hole in a LogicLock Region, Noncontiguous LogicLock Region, Routing Regions, Logic Placed Outside of an Empty Region. • Updated figures: HSSI Channel Blocks, Highlight Routing, High-Speed and Low Power Tiles in an Arria 10 Device, Show Delays Highlight Routing, Viewing Assignments in the Chip Planner, LogicLock Plus Regions Window, Using the Merge LogicLock Plus Region Command. • Created topics: *Adding Rectangle to a LogicLock Plus Region*, *Subtracting Rectangle from a LogicLock Plus Region*. • Moved topic: *Viewing Critical Paths* to *Timing Closure and Optimization* chapter and renamed to *Critical Paths*. • Renamed topic: *Creating Non-Rectangular LogicLock Plus Regions* to *Merging LogicLock Plus Regions*. • Renamed topic: *Chip Planner Overview* to *Design Floorplan Analysis in the Chip Planner*. • Renamed chapter from *Analyzing and Optimizing the Design Floorplan with the Chip Planner* to *Analyzing and Optimizing the Design Floorplan*.
2016.10.31	16.1.0	• Implemented Intel rebranding. • Added topic describing how to create a hole in a LogicLock Plus region.
2016.05.02	16.0.0	Updated information on creating LogicLock Plus regions.
2015.11.02	15.1.0	• Changed instances of *Quartus II* to *Quartus Prime*. • Added information on how to use LogicLock regions.

continued...

Date	Version	Changes
2015.05.04	15.0.0	Added information about color coding of LogicLock regions.
2014.12.15	14.1.0	Updated description of Virtual Pins assignment to clarify that assigned input is not available.
June 2014	14.0.0	Updated format
November 2013	13.1.0	Removed HardCopy device information.
May 2013	13.0.0	Updated "Viewing Routing Congestion" section Updated references to Quartus UI controls for the Chip Planner
June 2012	12.0.0	Removed survey link.
November 2011	11.0.1	Template update.
May 2011	11.0.0	• Updated for the 11.0 release. Edited "LogicLock Regions" Updated "Viewing Routing Congestion" Updated "Locate History" Updated Figures 15-4, 15-9, 15-10, and 15-13 Added Figure 15-6
December 2010	10.1.0	• Updated for the 10.1 release.
July 2010	10.0.0	• Updated device support information • Removed references to Timing Closure Floorplan; removed "Design Analysis Using the Timing Closure Floorplan" section • Added links to online Help topics • Added "Using LogicLock Regions with the Design Partition Planner" section • Updated "Viewing Critical Paths" section • Updated several graphics • Updated format of Document revision History table
November 2009	9.1.0	• Updated supported device information throughout • Removed deprecated sections related to the Timing Closure Floorplan for older device families. (For information on using the Timing Closure Floorplan with older device families, refer to previous versions of the Quartus Prime Handbook, available in the Altera Documentation Archive.) • Updated "Creating Nonrectangular LogicLock Regions" section • Added "Selected Elements Window" section • Updated table 12-1
May 2008	8.0.0	• Updated the following sections: "Chip Planner Tasks and Layers" "LogicLock Regions" "Back-Annotating LogicLock Regions" "LogicLock Regions in the Timing Closure Floorplan" • Added the following sections: "Reserve LogicLock Region" "Creating Nonrectangular LogicLock Regions" "Viewing Available Clock Networks in the Device" • Updated Table 10–1 • Removed the following sections: Reserve LogicLock Region Design Analysis Using the Timing Closure Floorplan

Related Links

Altera Documentation Archive

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

13 Netlist Optimizations and Physical Synthesis

13.1 Netlist Optimizations and Physical Synthesis

The Quartus Prime software offers netlist and physical synthesis optimizations that improve performance of your design. Click to enable physical synthesis options during fitting. This chapter also provides guidelines for applying netlist and physical synthesis options, and for preserving compilation results through back-annotation.

Table 54. Netlist Optimization and Physical Synthesis Options

Options	Location/Description
Enable physical synthesis options.	**Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Fitter)**. Physical synthesis optimizations apply at different stages of the compilation flow, either during synthesis, fitting, or both.
Enable netlist optimization options.	**Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Synthesis)**. Netlist optimizations operate with the atom netlist of your design, which describes a design in terms of specific primitives. An atom netlist file can be an Electronic Design Interchange Format (`.edf`) file or a Verilog Quartus Mapping (`.vqm`) file generated by a third-party synthesis tool. Quartus Prime synthesis generates and internally uses the atom netlist internally

Note: Because the node names for primitives in the design can change when you use physical synthesis optimizations, you should evaluate whether your design depends on fixed node names. If you use a verification flow that might require fixed node names, such as the Signal Tap Logic Analyzer, formal verification, or the LogicLock Plus based optimization flow (for legacy devices), disable physical synthesis options.

13.1.1 Physical Synthesis Optimizations

The Quartus Prime Fitter places and routes the logic cells to ensure critical portions of logic are close together and use the fastest possible routing resources. However, routing delays are often a significant part of the typical critical path delay. Physical synthesis optimizations take into consideration placement information, routing delays, and timing information to determine the optimal placement. The Fitter then focuses timing-driven optimizations at those critical parts of the design. The tight integration of the synthesis and fitting processes is known as physical synthesis.

To enable or disable physical synthesis options, click **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Fitter)**.

The following sections describe the physical synthesis optimizations available in the Quartus Prime software, and how they can help improve performance and fitting for the selected device.

Note: To disable global physical synthesis optimizations for specific elements of your design, assign the **Netlist Optimizations** logic option to **Never Allow** to the specific nodes or entities.

Related Links

Compiler Settings Page (Settings Dialog Box)

13.1.1.1 Enabling Physical Synthesis Optimization

Physical synthesis optimization improves circuit performance by performing combinational and sequential optimization and register duplication.

To enable physical synthesis options, follow these steps:

1. Click **Assignments ➤ Settings ➤ Compiler Settings**.
2. To enable retiming, combinational optimization, and register duplication, click **Advanced Settings (Fitter)**. Next, enable **Physical Synthesis**.
3. View physical synthesis results in the **Netlist Optimizations** report.

13.1.1.2 Physical Synthesis Options

The Quartus Prime software provides physical synthesis optimization options to improve fitting results. To access these options, click **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Fitter)**.

Table 55. Physical Synthesis Options

Option	Description
Advanced Physical Synthesis	Uses the physical synthesis engine to perform combinational and sequential optimization during fitting to improve circuit performance.
Netlist Optimizations	You can use the Assignment Editor to apply the **Netlist Optimizations** logic option. Use this option to disable physical synthesis optimizations for parts of your design.
Allow Register Duplication	Allows the Compiler to duplicate registers to improve design performance. When you enable this option, the Compiler copies registers and moves some fan-out to this new node. This optimization improves routability and can reduce the total routing wire in nets with many fan-outs. If you disable this option, this disables optimizations that retime registers. This setting affects Analysis & Synthesis and the Fitter.
Allow Register Merging	Allows the Compiler to remove registers that are identical to other registers in the design. When you enable this option, in cases where two registers generate the same logic, the Compiler deletes one register, and the remaining registers fan-out to the deleted register's destinations. This option is useful if you wish to prevent the Compiler from removing intentional use of duplicate registers. If you disable register merging, the Compiler disables optimizations that retime registers. This setting affects Analysis & Synthesis and the Fitter.

13.1.2 Applying Netlist Optimizations

The improvement in performance when using netlist optimizations is design dependent. If you have restructured your design to balance critical path delays, netlist optimizations might yield minimal improvement in performance.

You may have to experiment with available options to see which combination of settings works best for a particular design. Refer to the messages in the compilation report to see the magnitude of improvement with each option, and to help you decide whether you should turn on a given option or specific effort level.

Turning on more netlist optimization options can result in more changes to the node names in the design; bear this in mind if you are using a verification flow, such as the Signal Tap Logic Analyzer or formal verification that requires fixed or known node names.

To find the best results, you can use the Quartus Prime Design Space Explorer II (DSE) to apply various sets of netlist optimization options.

Related Links

- Design Space Explorer II on page 160
  Design Space Explorer II (DSE) is an easy-to-use, self-guided design optimization utility that is included in the Quartus Prime software.
- Optimizing with Design Space Explorer II
  In *Quartus Prime Pro Edition Handbook Volume 1: Design and Synthesis*

13.1.2.1 WYSIWYG Primitive Resynthesis

If you use a third-party tool to synthesize your design, use the **Perform WYSIWYG primitive resynthesis** option to apply optimizations to the synthesized netlist.

The **Perform WYSIWYG primitive resynthesis** option directs the Quartus Prime software to un-map the logic elements (LEs) in an atom netlist to logic gates, and then re-map the gates back to Intel-specific primitives. Third-party synthesis tools generate either an `.edf` or `.vqm` atom netlist file using Intel-specific primitives. When you turn on the **Perform WYSIWYG primitive resynthesis** option, the Quartus Prime software uses device-specific techniques during the re-mapping process. This feature re-maps the design using the **Optimization Technique** specified for your project (**Speed**, **Area**, or **Balanced**).

The **Perform WYSIWYG primitive resynthesis** option unmaps and remaps only logic cells, also referred to as LCELL or LE primitives, and regular I/O primitives (which may contain registers). Double data rate (DDR) I/O primitives, memory primitives, digital signal processing (DSP) primitives, and logic cells in carry/cascade chains are not remapped. This process does not process logic specified in an encrypted `.vqm` file or an `.edf` file, such as third-party intellectual property (IP).

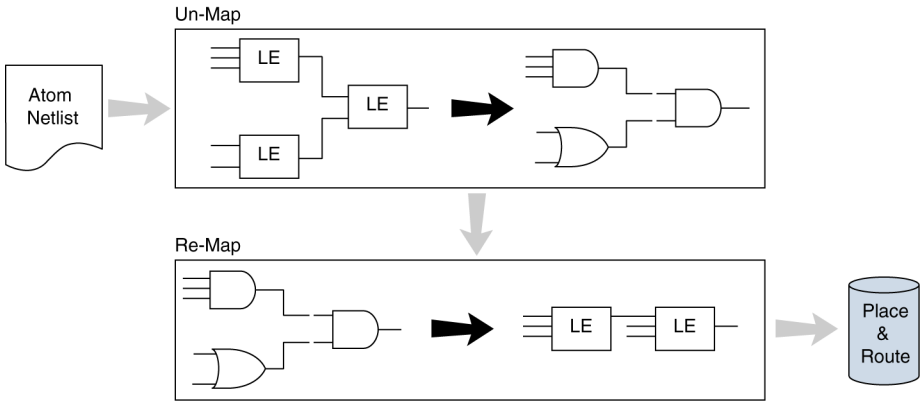
The **Perform WYSIWYG primitive resynthesis** option can change node names in the `.vqm` file or `.edf` file from your third-party synthesis tool, because the primitives in the atom netlist are broken apart and then re-mapped by the Quartus Prime software. The re-mapping process removes duplicate registers. Registers that are not removed retain the same name after re-mapping.

Any nodes or entities that have the **Netlist Optimizations** logic option set to **Never Allow** are not affected during WYSIWYG primitive resynthesis. You can use the Assignment Editor to apply the **Netlist Optimizations** logic option. This option disables WYSIWYG resynthesis for parts of your design.

Note: Primitive node names are specified during synthesis. When netlist optimizations are applied, node names might change because primitives are created and removed. HDL attributes applied to preserve logic in third-party synthesis tools cannot be maintained because those attributes are not written into the atom netlist, which the Quartus Prime software reads.

If you use the Quartus Prime software to synthesize your design, you can use the **Preserve Register (preserve)** and **Keep Combinational Logic (keep)** attributes to maintain certain nodes in the design.

Figure 89. Quartus Prime Flow for WYSIWYG Primitive Resynthesis



13.2 Scripting Support

You can run procedures and make settings described in this chapter in a Tcl script. You can also run some procedures at a command prompt. For detailed information about scripting command options, refer to the Quartus Prime Command-Line and Tcl API Help browser. To run the Help browser, type the following command at the command prompt:

```
quartus_sh --qhelp
```

You can specify many of the options described in this section on either an instance or global level, or both.

Use the following Tcl command to make a global assignment:

```
set_global_assignment -name <QSF variable name> <value>
```

Use the following Tcl command to make an instance assignment:

```
set_instance_assignment -name <QSF variable name> <value> \
-to <instance name>
```

Related Links

- Tcl Scripting on page 72
- API Functions for Tcl
  In Quartus Prime Help
- Command Line Scripting on page 65
  FPGA design software that easily integrates into your design flow saves time and improves productivity. The Intel Quartus Prime software provides you with a command-line executable for each step of the FPGA design flow to make the design process customizable and flexible.

- Quartus Prime Pro Edition Settings File Reference Manual
  For information about all settings and constraints in the Quartus Prime software.

13.2.1 Synthesis Netlist Optimizations

The project `.qsf` file preserves the settings that you specify in the GUI. Alternatively, you can edit the `.qsf` directly. The `.qsf` file supports the following synthesis netlist optimization commands. The **Type** column indicates whether the setting is supported as a global setting, an instance setting, or both.

Table 56. Synthesis Netlist Optimizations and Associated Settings

Setting Name	Quartus Prime Settings File Variable Name	Values	Type
Perform WYSIWYG Primitive Resynthesis	ADV_NETLIST_OPT_SYNTH_WYSIWYG_REMAP	ON, OFF	Global, Instance
Optimization Mode	OPTIMIZATION_MODE	BALANCEDHIGH PERFORMANCE EFFOR AGGRESSIVE PERFORMANCE	Global, Instance
Power-Up Don't Care	ALLOW_POWER_UP_DONT_CARE	ON, OFF	Global

13.2.2 Physical Synthesis Optimizations

The project `.qsf` file preserves the settings that you specify in the GUI. Alternatively, you can edit the `.qsf` directly. The `.qsf` file supports the following synthesis netlist optimization commands. The **Type** column indicates whether the setting is supported as a global setting, an instance setting, or both.

Table 57. Physical Synthesis Optimizations and Associated Settings

Setting Name	Quartus Prime Settings File Variable Name	Values	Type
Advanced Physical Synthesis	ADVANCED_PHYSICAL_SYNTHESIS	ON, OFF	Global

13.3 Document Revision History

Table 58. Document Revision History

Date	Version	Changes
2016.10.31	16.1.0	• Implemented Intel rebranding. • Updated physical synthesis options and procedure.
2016.05.02	16.0.0	• Removed information about deprecated physical synthesis options.
2015.11.02	15.1.0	• Changed instances of *Quartus II* to *Quartus Prime*. • Added Spectra-Q Physical Synthesis.
2014.12.15	14.1.0	• Updated location of Fitter Settings, Analysis & Synthesis Settings, and Physical Synthesis Optimizations Settings to Compiler Settings. • Updated DSE II content.
June 2014	14.0.0	Updated format.
November 2013	13.1.0	Removed HardCopy device information.

continued...

Date	Version	Changes
June 2012	12.0.0	Removed survey link.
November 2011	10.0.2	Template update.
December 2010	10.0.1	Template update.
July 2010	10.0.0	• Added links to Quartus Prime Help in several sections. • Removed Referenced Documents section. • Reformatted Document Revision History
November 2009	9.1.0	• Added information to "Physical Synthesis for Registers—Register Retiming" • Added information to "Applying Netlist Optimization Options" • Made minor editorial updates
March 2009	9.0.0	• Was chapter 11 in the 8.1.0 release. • Updated the "Physical Synthesis for Registers—Register Retiming" and "Physical Synthesis Options for Fitting" • Updated "Performing Physical Synthesis Optimizations" • Deleted Gate-Level Register Retiming section. • Updated the referenced documents
November 2008	8.1.0	Changed to 8½" × 11" page size. No change to content.
May 2008	8.0.0	• Updated "Physical Synthesis Optimizations for Performance on page 11-9 • Added Physical Synthesis Options for Fitting on page 11-16

Related Links

Altera Documentation Archive

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

14 Signal Integrity Analysis with Third-Party Tools

14.1 Signal Integrity Analysis with Third-Party Tools

With the ever-increasing operating speed of interfaces in traditional FPGA design, the timing and signal integrity margins between the FPGA and other devices on the board must be within specification and tolerance before a single PCB is built.

If the board trace is designed poorly or the route is too heavily loaded, noise in the signal can cause data corruption, while overshoot and undershoot can potentially damage input buffers over time.

As FPGA devices are used in high-speed applications, signal integrity and timing margin between the FPGA and other devices on the printed circuit board (PCB) are important aspects to consider to ensure proper system operation. To avoid time-consuming redesigns and expensive board respins, the topology and routing of critical signals must be simulated. The high-speed interfaces available on current FPGA devices must be modeled accurately and integrated into timing models and board-level signal integrity simulations. The tools used in the design of an FPGA and its integration into a PCB must be "board-aware"—able to take into account properties of the board routing and the connected devices on the board.

The Quartus Prime software provides methodologies, resources, and tools to ensure good signal integrity and timing margin between Intel FPGA devices and other components on the board. Three types of analysis are possible with the Quartus Prime software:

- I/O timing with a default or user-specified capacitive load and no signal integrity analysis (default)
- The Quartus Prime **Enable Advanced I/O Timing** option utilizing a user-defined board trace model to produce enhanced timing reports from accurate "board-aware" simulation models
- Full board routing simulation in third-party tools using Intel-provided or generated Input/Output Buffer Information Specification (IBIS) or HSPICE I/O models

I/O timing using a specified capacitive test load requires no special configuration other than setting the size of the load. I/O timing reports from the Quartus Prime TimeQuest or the Quartus Prime Classic Timing Analyzer are generated based only on point-to-point delays within the I/O buffer and assume the presence of the capacitive test load with no other details about the board specified. The default size of the load is based on the I/O standard selected for the pin. Timing is measured to the FPGA pin with no signal integrity analysis details.

The **Enable Advanced I/O Timing** option expands the details in I/O timing reports by taking board topology and termination components into account. A complete point-to-point board trace model is defined and accounted for in the timing analysis. This ability to define a board trace model is an example of how the Quartus Prime software is "board-aware."

In this case, timing and signal integrity metrics between the I/O buffer and the defined far end load are analyzed and reported in enhanced reports generated by the Quartus Prime TimeQuest Timing Analyzer.

Related Links

I/O Management on page 18
For more information about defining capacitive test loads or how to use the **Enable Advanced I/O Timing** option to configure a board trace model.

14.1.1 Signal Integrity Simulations with HSPICE and IBIS Models

The Quartus Prime software can export accurate HSPICE models with the built-in HSPICE Writer. You can run signal integrity simulations with these complete HSPICE models in Synopsys HSPICE. IBIS models of the FPGA I/O buffers are also created easily with the Quartus Prime IBIS Writer.

You can run signal integrity simulations with these complete HSPICE models in Synopsys HSPICE.

You can integrate IBIS models into any third-party simulation tool that supports them, such as the Mentor Graphics ® Hyperlynx software. With the ability to create industry-standard model definition files quickly, you can build accurate simulations that can provide data to help improve board-level signal integrity.

The I/O's IBIS and HSPICE model creation available in the Quartus Prime software can help prevent problems before a costly board respin is required. In general, creating and running accurate simulations is difficult and time consuming. The tools in the Quartus Prime software automate the I/O model setup and creation process by configuring the models specifically for your design. With these tools, you can set up and run accurate simulations quickly and acquire data that helps guide your FPGA and board design.

The information about signal integrity in this chapter refers to board-level signal integrity based on I/O buffer configuration and board parameters, not simultaneous switching noise (SSN), also known as ground bounce or V_{CC} sag. SSN is a product of multiple output drivers switching at the same time, causing an overall drop in the voltage of the chip's power supply. This can cause temporary glitches in the specified level of ground or V_{CC} for the device.

This chapter is intended for FPGA and board designers and includes details about the concepts and steps involved in getting designs simulated and how to adjust designs to improve board-level timing and signal integrity. Also included is information about how to create accurate models from the Quartus Prime software and how to use those models in simulation software.

The information in this chapter is meant for those who are familiar with the Quartus Prime software and basic concepts of signal integrity and the design techniques and components in good PCB design. Finally, you should know how to set up simulations and use your selected third-party simulation tool.

Related Links

- AN 315: Guidelines for Designing High-Speed FPGA PCBs
  For a more information about SSN and ways to prevent it.
- Altera Signal Integrity Center

For information about basic signal integrity concepts and signal integrity details pertaining to Intel FPGA devices.

14.2 I/O Model Selection: IBIS or HSPICE

The Quartus Prime software can export two different types of I/O models that are useful for different simulation situations, IBIS models and HSPICE models.

IBIS models define the behavior of input or output buffers through the use of voltage-current (V-I) and voltage-time (V-t) data tables. HSPICE models, often referred to as HSPICE decks, include complete physical descriptions of the transistors and parasitic capacitances that make up an I/O buffer along with all the parameter settings required to run a simulation. The HSPICE decks generated by the Quartus Prime software are preconfigured with the I/O standard, voltage, and pin loading settings for each pin in your design.

The choice of I/O model type is based on many factors.

Table 59. IBIS and HSPICE Model Comparison

Feature	IBIS Model	HSPICE Model
I/O Buffer Description	**Behavioral**—I/O buffers are described by voltage-current and voltage-time tables in typical, minimum, and maximum supply voltage cases.	**Physical**—I/O buffers and all components in a circuit are described by their physical properties, such as transistor characteristics and parasitic capacitances, as well as their connections to one another.
Model Customization	**Simple and limited**—The model completely describes the I/O buffer and does not usually have to be customized.	**Fully customizable**—Unless connected to an arbitrary board description, the description of the board trace model must be customized in the model file. All parameters of the simulation are also adjustable.
Simulation Set Up and Run Time	**Fast**—Simulations run quickly after set up correctly.	**Slow**—Simulations take time to set up and take longer to run and complete.
Simulation Accuracy	**Good**—For most simulations, accuracy is sufficient to make useful adjustments to the FPGA and/or board design to improve signal integrity.	**Excellent**—Simulations are highly accurate, making HSPICE simulation almost a requirement for any high-speed design where signal integrity and timing margins are tight.
Third-Party Tool Support	**Excellent**—Almost all third-party board simulation tools support IBIS.	**Good**—Most third-party tools that support SPICE support HSPICE. However, Synopsys HSPICE is required for simulations of Intel's encrypted HSPICE models.

Related Links

AN 283: Simulating Intel Devices with IBIS Models
For more information about IBIS files created by the Quartus Prime IBIS Writer and IBIS files in general, as well as links to websites with detailed information.

14.3 FPGA to Board Signal Integrity Analysis Flow

Board signal integrity analysis can take place at any point in the FPGA design process and is often performed before and after board layout. If it is performed early in the process as part of a pre-PCB layout analysis, the models used for simulations can be more generic.

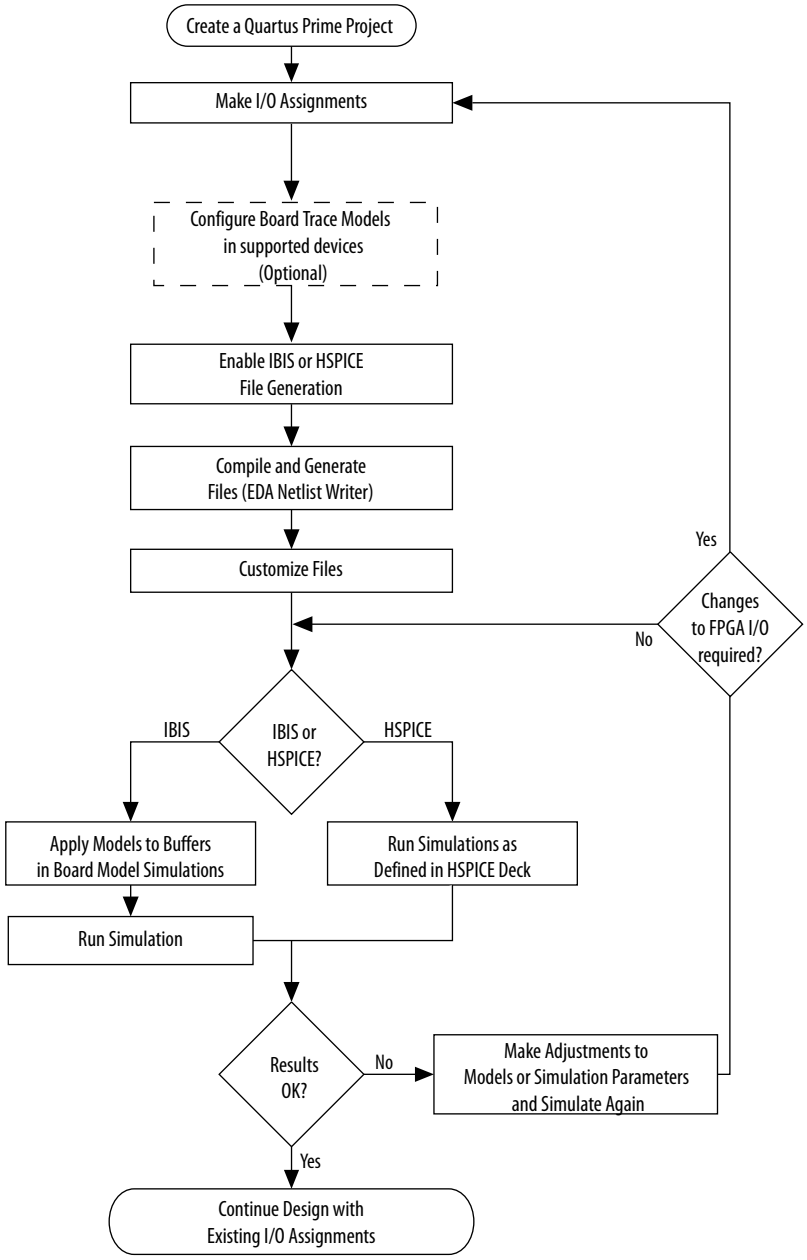
These models can be changed as much as required to see how adjustments improve timing or signal integrity and help with the design and routing of the PCB. Simulations and the resulting changes made at this stage allow you to analyze "what if" scenarios to plan and implement your design better. To assist with early board signal integrity

analysis, you can download generic IBIS model files for each device family and obtain HSPICE buffer simulation kits from the "Board Level Tools" section of the EDA Tool Support Resource Center.

Typically, if board signal integrity analysis is performed late in the design, it is used for a post-layout verification. The inputs and outputs of the FPGA are defined, and required board routing topologies and constraints are known. Simulations can help you find problems that might still exist in the FPGA or board design before fabrication and assembly. In either case, a simple process flow illustrates how to create accurate IBIS and HSPICE models from a design in the Quartus Prime software and transfer them to third-party simulation tools.

Your design depends on the type of model, IBIS or HSPICE, that you use for your simulations. When you understand the steps in the analysis flow, refer to the section of this chapter that corresponds to the model type you are using.

Figure 90. Third-Party Board Signal Integrity Analysis Flow



Related Links

EDA Tool Support Resource Center

For more information, generic IBIS model files for each device family, and to obtain HSPICE buffer simulation kits.

14.3.1 Create I/O and Board Trace Model Assignments

You can configure a board trace model for output signals or for bidirectional signals in output mode. You can then automatically transfer its description to HSPICE decks generated by the HSPICE Writer. This helps improve simulation accuracy.

To configure a board trace model, in the **Settings** dialog box, in the **TimeQuest Timing Analyzer** page, turn on the **Enable Advanced I/O Timing** option and configure the board trace model assignment settings for each I/O standard used in your design. You can add series or parallel termination, specify the transmission line length, and set the value of the far-end capacitive load. You can configure these parameters either in the Board Trace Model view of the Pin Planner, or click **SettingsDeviceDevice and Pin Options**.

The Quartus Prime software can generate IBIS models and HSPICE decks without having to configure a board trace model with the **Enable Advanced I/O Timing** option. In fact, IBIS models ignore any board trace model settings other than the far-end capacitive load. If any load value is set other than the default, the delay given by IBIS models generated by the IBIS Writer cannot be used to account correctly for the double counting problem. The load value mismatch between the IBIS delay and the t_{CO} measurement of the Quartus Prime software prevents the delays from being safely added together. Warning messages displayed when the EDA Netlist Writer runs indicate when this mismatch occurs.

Related Links

I/O Management on page 18
For information about how to use the**Enable Advanced I/O Timing** option and configure board trace models for the I/O standards used in your design.

14.3.2 Output File Generation

IBIS and HSPICE model files are not generated by the Quartus Prime software by default. To generate or update the files automatically during each project compilation, select the type of file to generate and a location where to save the file in the project settings.

The IBIS and HSPICE Writers in the Quartus Prime software are run as part of the EDA Netlist Writer during normal project compilation. If either writer is turned on in the project settings, IBIS or HSPICE files are created and stored in the specified location. For IBIS, a single file is generated containing information about all assigned pins. HSPICE file generation creates separate files for each assigned pin. You can run the EDA Netlist Writer separately from a full compilation in the Quartus Prime software or at the command line.

14.3.3 Customize the Output Files

The files generated by either the IBIS or HSPICE Writer are text files that you can edit and customize easily for design or experimentation purposes.

IBIS files downloaded from the Altera website must be customized with the correct RLC values for the specific device package you have selected for your design. IBIS files generated by the IBIS Writer do not require this customization because they are configured automatically with the RLC values for your selected device. HSPICE decks require modification to include a detailed description of your board. With **Enable Advanced I/O Timing** turned on and a board trace model defined in the Quartus

Prime software, generated HSPICE decks automatically include that model's parameters. However, Intel recommends that you replace that model with a more detailed model that describes your board design more accurately. A default simulation included in the generated HSPICE decks measures delay between the FPGA and the far-end device. You can make additions or adjustments to the default simulation in the generated files to change the parameters of the default simulation or to perform additional measurements.

14.3.4 Set Up and Run Simulations in Third-Party Tools

When you have generated the files, you can use them to perform simulations in your selected simulation tool.

With IBIS models, you can apply them to input, output, or bidirectional buffer entities and quickly set up and run simulations. For HSPICE decks, the simulation parameters are included in the files. Open the files in Synopsys HSPICE and run simulations for each pin as required.

With HSPICE decks generated from the HSPICE Writer, the double counting problem is accounted for, which ensures that your simulations are accurate. Simulations that involve IBIS models created with anything other than the default loading settings in the Quartus Prime software must take the change in the size of the load between the IBIS delay and the Quartus Prime t_{CO} measurement into account. Warning messages during compilation alert you to this change.

14.3.5 Interpret Simulation Results

If you encounter timing or signal integrity issues with your high-speed signals after running simulations, you can make adjustments to I/O assignment settings in the Quartus Prime software.

These could include such things as drive strength or I/O standard, or making changes to your board routing or topology. After regenerating models in the Quartus Prime software based on the changes you have made, rerun the simulations to check whether your changes corrected the problem.

14.4 Simulation with IBIS Models

IBIS models provide a way to run accurate signal integrity simulations quickly. IBIS models describe the behavior of I/O buffers with voltage-current and voltage-time data curves.

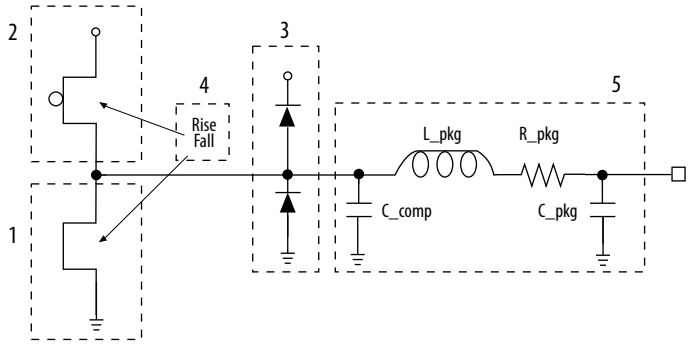
Because of their behavioral nature, IBIS models do not have to include any information about the internal circuit design of the I/O buffer. Most component manufacturers, including Intel, provide IBIS models for free download and use in signal integrity analysis simulation tools. You can download generic device family IBIS models from the Altera website for early design simulation or use the IBIS Writer to create custom IBIS models for your existing design.

14.4.1 Elements of an IBIS Model

An IBIS model file (**`.ibs`**) is a text file that describes the behavior of an I/O buffer across minimum, typical, and maximum temperature and voltage ranges with a specified test load.

The tables and values specified in the IBIS file describe five basic elements of the I/O buffer.

Figure 91. Five Basic Elements of an I/O Buffer in IBIS Models



The following elements correspond to each numbered block.

1. **Pulldown**—A voltage-current table describes the current when the buffer is driven low based on a pull-down voltage range of $-V_{CC}$ to 2 V_{CC}.
2. **Pullup**—A voltage-current table describes the current when the buffer is driven high based on a pull-up voltage range of $-V_{CC}$ to V_{CC}.
3. **Ground and Power Clamps**—Voltage-current tables describe the current when clamping diodes for electrostatic discharge (ESD) are present. The ground clamp voltage range is $-V_{CC}$ to V_{CC}, and the power clamp voltage range is $-V_{CC}$ to ground.
4. **Ramp and Rising/Falling Waveform**—A voltage-time (dv/dt) ratio describes the rise and fall time of the buffer during a logic transition. Optional rising and falling waveform tables can be added to more accurately describe the characteristics of the rising and falling transitions.
5. **Total Output Capacitance and Package RLC**—The total output capacitance includes the parasitic capacitances of the output pad, clamp diodes (if present), and input transistors. The package RLC is device package-specific and defines the resistance, inductance, and capacitance of the bond wire and pin of the I/O.

Related Links

AN 283: Simulating Intel Devices with IBIS Models
For more information about IBIS models and Intel-specific features, including links to the official IBIS specification.

14.4.2 Creating Accurate IBIS Models

There are two methods to obtain Intel device IBIS files for your board-level signal integrity simulations. You can download generic IBIS models from the Altera website. You can also use the IBIS writer in the Quartus Prime software to create design-specific models.

The IBIS file generated by the Quartus Prime software contains models of both input and output termination, and is supported for IBIS model versions of 4.2 and later. Arria V , Cyclone V , and Stratix V device families allow the use of bidirectional I/O with dynamic on-chip termination (OCT).

Dynamic OCT is used where a signal uses a series on-chip termination during output operation and a parallel on-chip termination during input operation. Typically this is used in Altera External Memory Interface IP.

The Quartus Prime IBIS dynamic OCT IBIS model names end in `g50c_r50c`. For example : `sstl15i_ctnio_g50c_r50c`.

In the simulation tool, the IBIS model is attached to a buffer.

- When the buffer is assigned as an output, use the series termination r50c.
- When the buffer is assigned as an input, use the parallel termination g50c.

14.4.2.1 Download IBIS Models

Intel provides IBIS models for almost all FPGA and FPGA configuration devices. You can use the IBIS models from the website to perform early simulations of the I/O buffers you expect to use in your design as part of a pre-layout analysis.

Downloaded IBIS models have the RLC package values set to one particular device in each device family.

The **`.ibs`** file can be customized for your device package and can be used for any simulation. IBIS models downloaded and used for simulations in this manner are generic. They describe only a certain set of models listed for each device on the Intel IBIS Models page of the Altera website. To create customized models for your design, use the IBIS Writer as described in the next section.

To simulate your design with the model accurately, you must adjust the RLC values in the IBIS model file to match the values for your particular device package by performing the following steps:

1. Download and expand the ZIP file (`.zip`) of the IBIS model for the device family you are using for your design. The `.zip` file contains the `.ibs` file along with an IBIS model user guide and a model data correlation report.
2. Download the Package RLC Values spreadsheet for the same device family.
3. Open the spreadsheet and locate the row that describes the device package used in your design.
4. From the package's **I/O** row, copy the minimum, maximum, and typical values of resistance, inductance, and capacitance for your device package.
5. Open the `.ibs` file in a text editor and locate the [Package] section of the file.
6. Overwrite the listed values copied with the values from the spreadsheet and save the file.

Related Links

Intel IBIS Models
For information about whether models for your selected device are available.

14.4.2.2 Generate Custom IBIS Models with the IBIS Writer

If you have started your FPGA design and have created custom I/O assignments, you can use the Quartus Prime IBIS Writer to create custom IBIS models to accurately reflect your assignments.

Examples of custom assignments include drive strength settings or the enabling of clamping diodes for ESD protection. IBIS models created with the IBIS Writer take I/O assignment settings into account.

If the **Enable Advanced I/O Timing** option is turned off, the generated **.ibs** files are based on the load value setting for each I/O standard on the **Capacitive Loading** page of the **Device and Pin Options** dialog box in the **Device** dialog box. With the **Enable Advanced I/O Timing** option turned on, IBIS models use an effective capacitive load based on settings found in the board trace model on the **Board Trace Model** page in the **Device and Pin Options** dialog box or the **Board Trace Model** view in the Pin Planner. The effective capacitive load is based on the sum of the **Near capacitance**, **Transmission line distributed capacitance**, and the **Far capacitance** settings in the board trace model. Resistance values and transmission line inductance values are ignored.

Note: If you made any changes from the default load settings, the delay in the generated IBIS model cannot safely be added to the Quartus Prime t_{CO} measurement to account for the double counting problem. This is because the load values between the two delay measurements do not match. When this happens, the Quartus Prime software displays warning messages when the EDA Netlist Writer runs to remind you about the load value mismatch.

Related Links

- Intel IBIS models
- Generating IBIS Output Files with the Quartus Prime Software
  in Quartus Prime Help
- AN 283: Simulating Intel Devices with IBIS Models

14.4.3 Design Simulation Using the Mentor Graphics HyperLynx® Software

You must integrate IBIS models downloaded from the Altera website or created with the Quartus Prime IBIS Writer into board design simulations to accurately model timing and signal integrity.

The HyperLynx software from Mentor Graphics is one of the most popular tools for design simulation. The HyperLynx software makes it easy to integrate IBIS models into simulations.

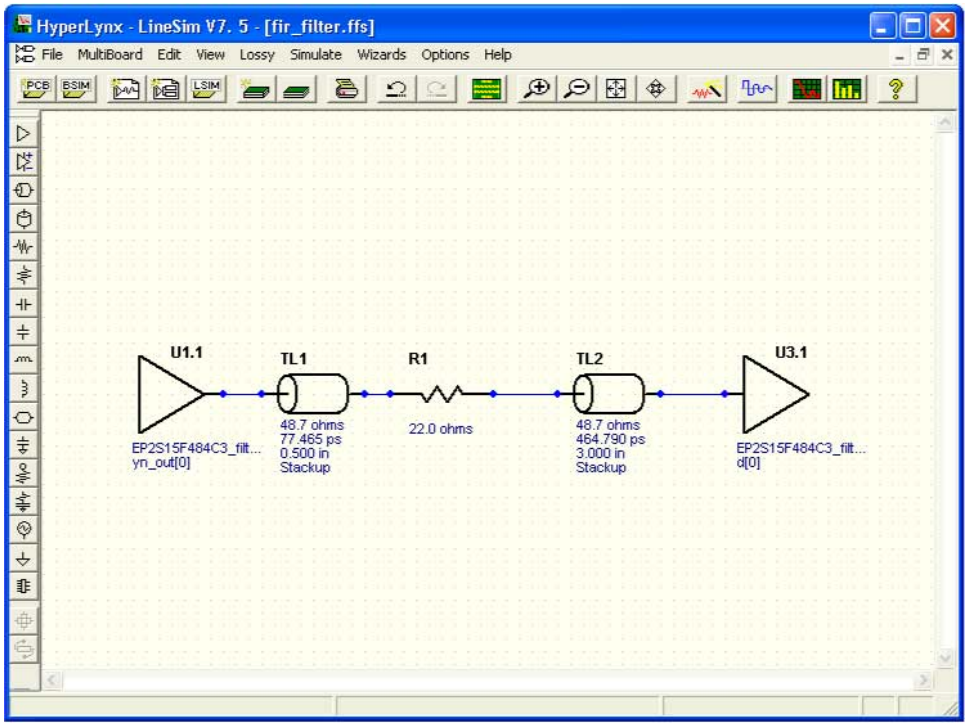
The HyperLynx software is a PCB analysis and simulation tool for high-speed designs, consisting of two products, LineSim and BoardSim. LineSim is an early simulation tool. Before any board routing takes place, LineSim is used to simulate "what if" scenarios to assist in creating routing rules and defining board parameters. BoardSim is a post-layout tool used to analyze existing board routing. Specific nets are selected from a board layout file and simulated in a manner similar to LineSim. With board and routing parameters, and surrounding signal routing known, highly accurate simulations of the final fabricated PCB are possible. This section focuses on LineSim. Because the process of creating and running simulations is very similar for both LineSim and BoardSim, the details of IBIS model use in LineSim applies to simulations in BoardSim.

Simulations in LineSim are configured using a schematic GUI to create connections and topologies between I/O buffers, route trace segments, and termination components. LineSim provides two methods for creating routing schematics: cell-based and free-form. Cell-based schematics are based on fixed cells consisting of

typical placements of buffers, trace impedances, and components. Parts of the grid-based cells are filled with the desired objects to create the topology. A topology in a cell-based schematic is limited by the available connections within and between the cells.

A more robust and expandable way to create a circuit schematic for simulation is to use the free-form schematic format in LineSim. The free-form schematic format makes it easy to place parts into any configuration and edit them as required. This section describes the use of IBIS models with free-form schematics, but the process is nearly identical for cell-based schematics.

Figure 92. HyperLynx LineSim Free-Form Schematic Editor



When you use HyperLynx software to perform simulations, you typically perform the following steps:

1. Create a new LineSim free-form schematic document and set up the board stackup for your PCB using the Stackup Editor. In this editor, specify board layer properties including layer thickness, dielectric constant, and trace width.
2. Create a circuit schematic for the net you want to simulate. The schematic represents all the parts of the routed net including source and destination I/O buffers, termination components, transmission line segments, and representations of impedance discontinuities such as vias or connectors.
3. Assign IBIS models to the source and destination I/O buffers to represent their behavior during operation.
4. Attach probes from the digital oscilloscope that is built in to LineSim to points in the circuit that you want to monitor during simulation. Typically, at least one probe is attached to the pin of a destination I/O buffer. For differential signals, you can attach a differential probe to both the positive and negative pins at the destination.
5. Configure and run the simulation. You can simulate a rising or falling edge and test the circuit under different drive strength conditions.
6. Interpret the results and make adjustments. Based on the waveforms captured in the digital oscilloscope, you can adjust anything in the circuit schematic to correct any signal integrity issues, such as overshoot or ringing. If necessary, you can make I/O assignment changes in the Quartus Prime software, regenerate the IBIS file with the IBIS Writer, and apply the updated IBIS model to the buffers in your HyperLynx software schematic.
7. Repeat the simulations and circuit adjustments until you are satisfied with the results. When the operation of the net meets your design requirements, implement changes to your I/O assignments in the Quartus Prime software and/or adjust your board routing constraints, component values, and placement to match the simulation.

Related Links

www.mentor.com
For more information about HyperLynx software, including schematic creation, simulation setup, model usage, product support, licensing, and training.

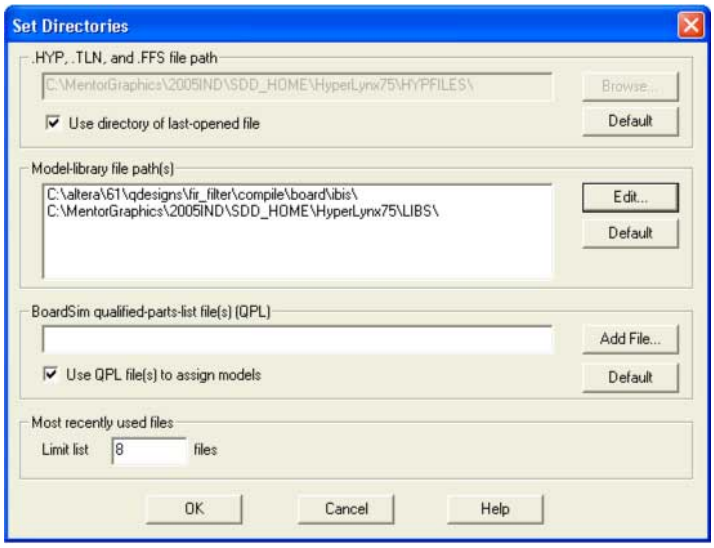
14.4.4 Configuring LineSim to Use Intel IBIS Models

You must configure LineSim to find and use the downloaded or generated IBIS models for your design. To do this, add the location of your **.ibs** file or files to the LineSim Model Library search path. Then you apply a selected model to a buffer in your schematic.

To add the Quartus Prime software's default IBIS model location, *<project directory>***/board/ibis**, to the HyperLynx LineSim model library search path, perform the following steps in LineSim:

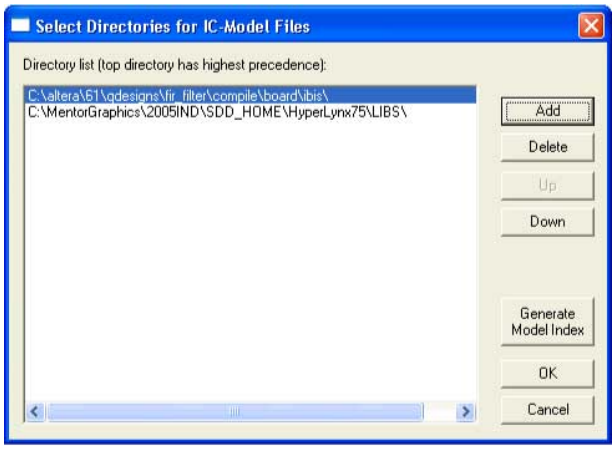
1. From the Options menu, click **Directories**. The **Set Directories** dialog box appears. The **Model-library file path(s)** list displays the order in which LineSim searches file directories for model files.

Figure 93. LineSim Set Directories Dialog Box



2. Click **Edit**. A dialog box appears where you can add directories and adjust the order in which LineSim searches them.

Figure 94. LineSim Select Directories Dialog Box



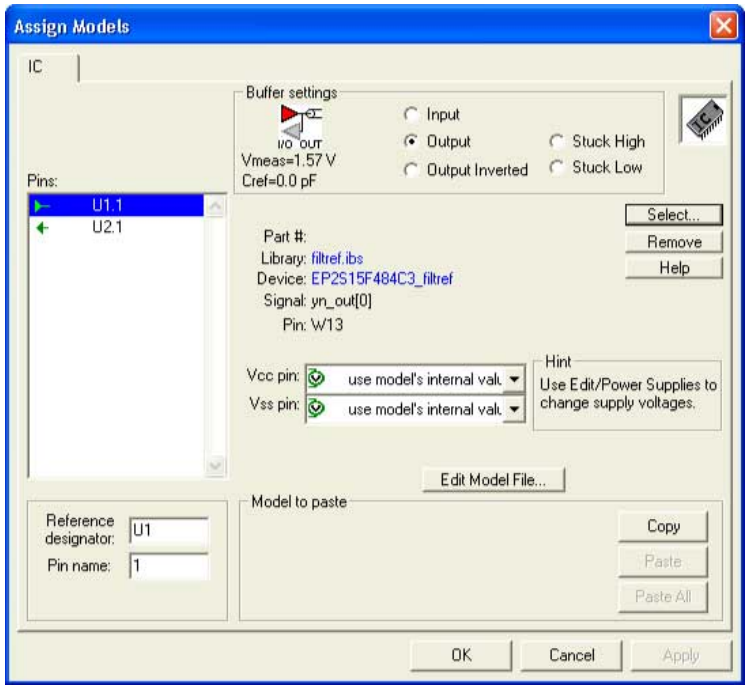
3. Click **Add**
4. Browse to the default IBIS model location, <*project directory*>**/board/ibis**. Click **OK**.
5. Click **Up** to move the IBIS model directory to the top of the list. Click **Generate Model Index** to update LineSim's model database with the models found in the added directory.
6. Click **OK**. The IBIS model directory for your project is added to the top of the Model-library file path(s) list.
7. To close the **Set Directories** dialog box, click **OK**.

14.4.5 Integrating Intel IBIS Models into LineSim Simulations

When the location for IBIS files has been set, you can assign the downloaded or generated IBIS models to the buffers in your schematic. To do this, perform the following steps:

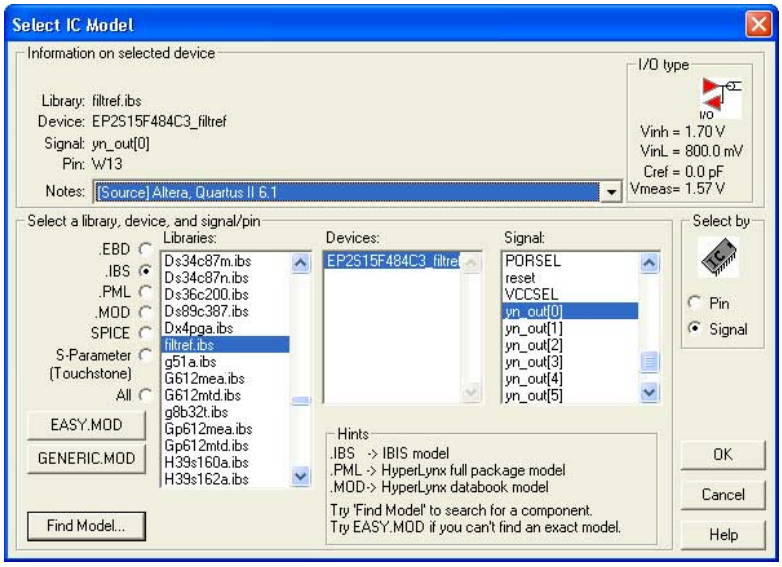
1. Double-click a buffer symbol in your schematic to open the **Assign Models** dialog box. You can also click **Assign Models** from the buffer symbol's right-click menu.

Figure 95. LineSim Assign Model Dialog Box



2. The pin of the buffer symbol you selected should be highlighted in the **Pins** list. If you want to assign a model to a different symbol or pin, select it from the list.
3. Click **Select**. The **Select IC Model** dialog box appears.

Figure 96. LineSim Select IC Model Dialog Box



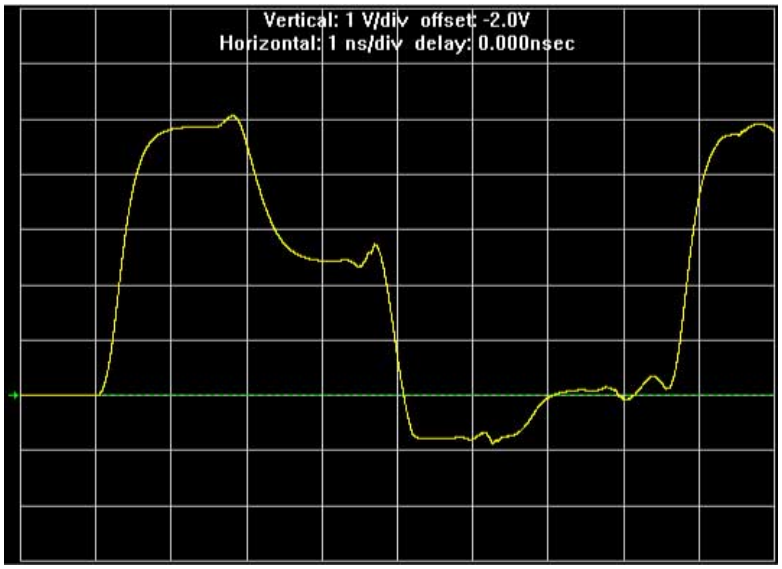
4. To filter the list of available libraries to display only IBIS models, select **.IBS**. Scroll through the **Libraries** list, and click the name of the library for your design. By default, this is <*project name*>**.ibs**.
5. The device for your design should be selected as the only item in the **Devices** list. If not, select your device from the list.
6. From the **Signal** list, select the name of the signal you want to simulate. You can also choose to select by device pin number.
7. Click **OK**. The **Assign Models** dialog box displays the selected **.ibs** file and signal.
8. If applicable to the signal you chose, adjust the buffer settings as required for the simulation.
9. Select and configure other buffer pins from the **Pins** list in the same manner.
10. Click **OK** when all I/O models are assigned.

14.4.6 Running and Interpreting LineSim Simulations

You can now run any desired simulations and make adjustments to the I/O assignments or simulation parameters as required.

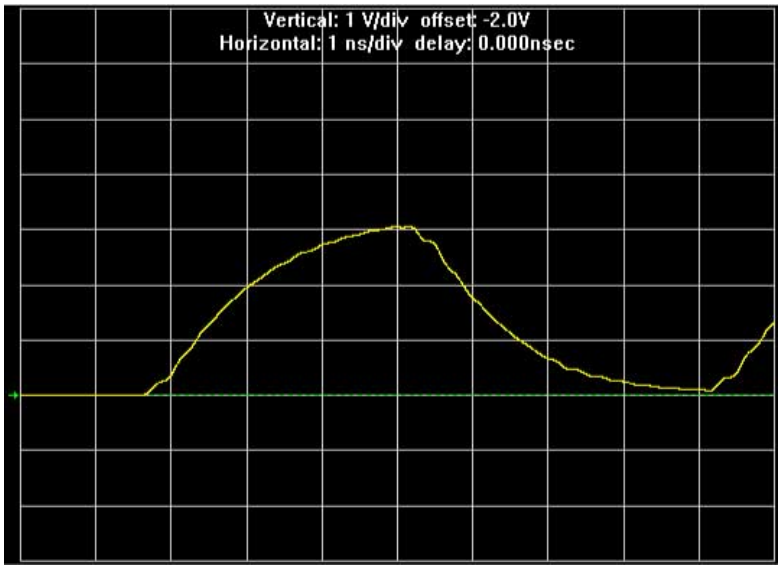
For example, if you see too much overshoot in the simulated signal at the destination buffer after running a simulation, you could adjust the drive strength I/O assignment setting to a lower value. Regenerate the **.ibs** file, and run the simulation again to verify whether the change fixed the problem.

Figure 97. Example of Overshoot in HyperLynx with IBIS Models



If you see a discontinuity or other anomalies at the destination, such as slow rise and fall times, adjust the termination scheme or termination component values. After making these changes, rerun the simulation to check whether your adjustments solved the problem. In this case, it is not necessary to regenerate the **.ibs** file.

Figure 98. Example of Signal Integrity Anomaly in HyperLynx with IBIS Models



Related Links

Altera Signal Integrity Center

For more information about board-level signal integrity and to learn about ways to improve it with simple changes to your design.

14.5 Simulation with HSPICE Models

HSPICE decks are used to perform highly accurate simulations by describing the physical properties of all aspects of a circuit precisely. HSPICE decks describe I/O buffers, board components, and all of the connections between them, as well as defining the parameters of the simulation to be run.

By their nature, HSPICE decks are highly customizable and require a detailed description of the circuit under simulation. For devices that support advanced I/O timing, when **Enable Advanced I/O Timing** is turned on, the HSPICE decks generated by the Quartus Prime HSPICE Writer automatically include board components and topology defined in the Board Trace Model. Configure the board components and topology in the Pin Planner or in the **Board Trace Model** tab of the **Device and Pin Options** dialog box. All HSPICE decks generated by the Quartus Prime software include compensation for the double count problem. You can simulate with the default simulation parameters built in to the generated HSPICE decks or make adjustments to customize your simulation.

Related Links

The Double Counting Problem in HSPICE Simulations on page 238
Simulating I/Os using accurate models is extremely helpful for finding and fixing FPGA I/O timing and board signal integrity issues before any boards are built. However, the usefulness of such simulations is directly related to the accuracy of the models used and whether the simulations are set up and performed correctly. To ensure accuracy in models and simulations created for FPGA output signals, the timing hand-off between t_{CO} timing in the Quartus Prime software and simulation-based board delay must be taken into account. If this hand-off is not handled correctly, the calculated delay could either count some of the delay twice or even miss counting some of the delay entirely.

14.5.1 Supported Devices and Signaling

The HSPICE Writer in the Quartus Prime software supports Arria , Cyclone, and Stratix devices for the creation of a board trace model in the Quartus Prime software for automatic inclusion in an HSPICE deck.

The HSPICE files include the board trace description you create in the Board Trace Model view in the Pin Planner or the **Board Trace Model** tab in the **Device and Pin Options** dialog box.

Note: Note that for Arria 10 devices, you may need to download the Encrypted HSPICE model from the Altera website.

Related Links

- I/O Management on page 18
  For more information about the **Enable Advanced I/O Timing** option and configuring board trace models for the I/O standards in your design.
- SPICE Models for Intel Devices
  For more information about the Encrypted HSPICE model.

14.5.2 Accessing HSPICE Simulation Kits

You can access the available HSPICE models with the Quartus Prime software's HSPICE Writer tool and also at the Spice Models for Intel Devices web page.

The Quartus Prime software HSPICE Writer tool removes many common sources of user error from the I/O simulation process. The HSPICE Writer tool automatically creates preconfigured I/O simulation spice decks that only require the addition of a user board model. All the difficult tasks required to configure the I/O modes and interpret the timing results are handled automatically by the HSPICE Writer tool.

Related Links

Spice Models for Intel Devices
For more information about downloadable HSPICE models.

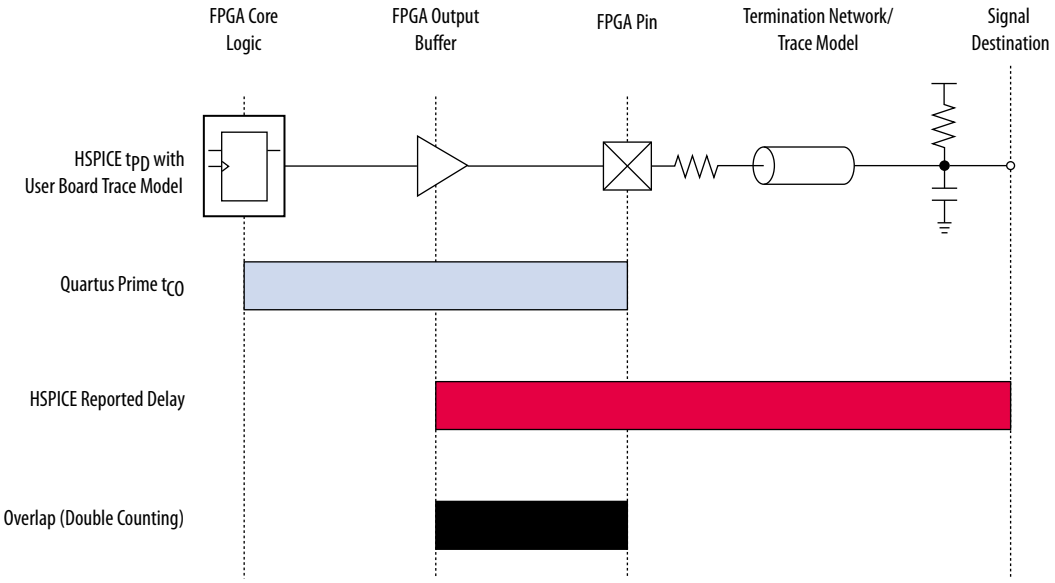
14.5.3 The Double Counting Problem in HSPICE Simulations

Simulating I/Os using accurate models is extremely helpful for finding and fixing FPGA I/O timing and board signal integrity issues before any boards are built. However, the usefulness of such simulations is directly related to the accuracy of the models used and whether the simulations are set up and performed correctly. To ensure accuracy in models and simulations created for FPGA output signals, the timing hand-off between t_{CO} timing in the Quartus Prime software and simulation-based board delay must be taken into account. If this hand-off is not handled correctly, the calculated delay could either count some of the delay twice or even miss counting some of the delay entirely.

14.5.3.1 Defining the Double Counting Problem

The double counting problem is inherent to the method output timing is analyzed versus the method used for HSPICE models. The timing analyzer tools in the Quartus Prime software measure delay timing for an output signal from the core logic of the FPGA design through the output buffer ending at the FPGA pin with a default capacitive load or a specified value for the selected I/O standard. This measurement is the t_{CO} timing variable.

Figure 99. Double Counting Problem



HSPICE models for board simulation measure t_{PD} (propagation delay) from an arbitrary reference point in the output buffer, through the device pin, out along the board routing, and ending at the signal destination.

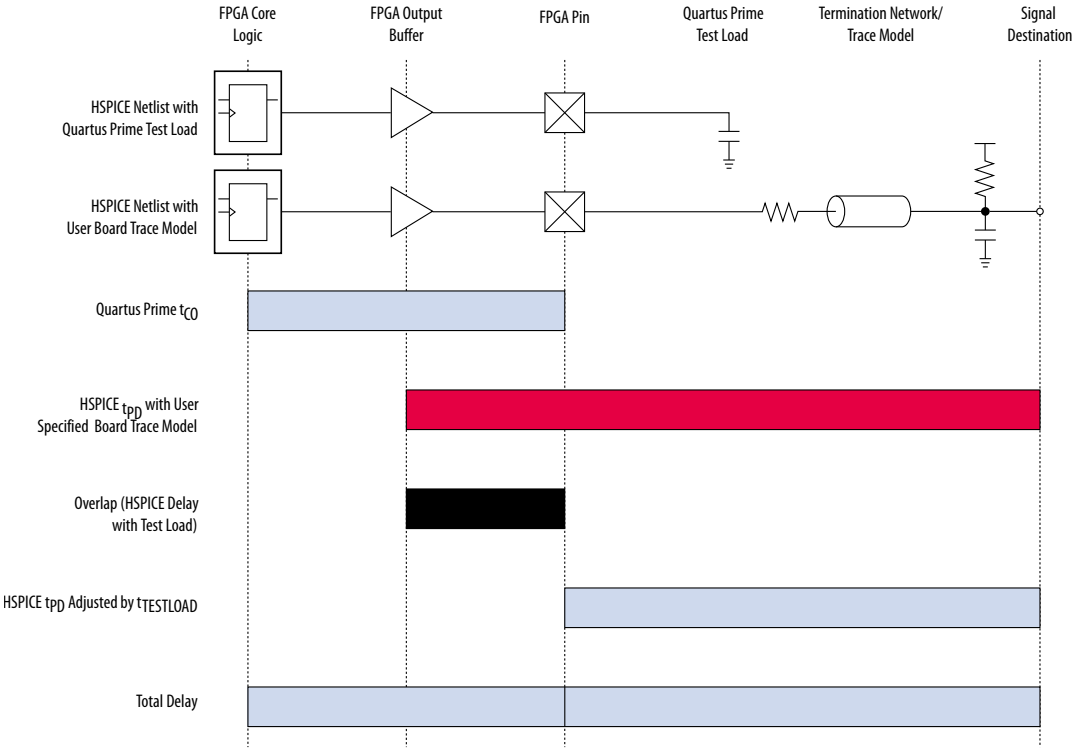
It is apparent immediately that if these two delays were simply added together, the delay between the output buffer and the device pin would be counted twice in the calculation. A model or simulation that does not account for this double count would create overly pessimistic simulation results, because the double-counted delay can limit I/O performance artificially. To fix the problem, it might seem that simply subtracting the overlap between t_{CO} and t_{PD} would account for the double count. However, this adjustment would not be accurate because each measurement is based on a different load.

Note: Input signals do not exhibit this problem because the HSPICE models for inputs stop at the FPGA pin instead of at the input buffer. In this case, simply adding the delays together produces an accurate measurement of delay timing.

14.5.3.2 The Solution to Double Counting

To adjust the measurements to account for the double-counting, the delay between the arbitrary point in the output buffer selected by the HSPICE model and the FPGA pin must be subtracted from either t_{CO} or t_{PD} before adding the results together. The subtracted delay must also be based on a common load between the two measurements. This is done by repeating the HSPICE model measurement, but with the same load used by the Quartus Prime software for the t_{CO} measurement.

Figure 100. Common Test Loads Used for Output Timing



With $t_{TESTLOAD}$ known, the total delay is calculated for the output signal from the FPGA logic to the signal destination on the board, accounting for the double count.

$t_{delay} = t_{CO}+(t_{PD}-t_{TESTLOAD})$

The preconfigured simulation files generated by the HSPICE Writer in the Quartus Prime software are designed to account for the double-counting problem based on this calculation automatically.

14.5.4 HSPICE Writer Tool Flow

This section includes information to help you get started using the Quartus Prime software HSPICE Writer tool. The information in this section assumes you have a basic knowledge of the standard Quartus Prime software design flow, such as project and assignment creation, compilation, and timing analysis.

14.5.4.1 Applying I/O Assignments

The first step in the HSPICE Writer tool flow is to configure the I/O standards and modes for each of the pins in your design properly. In the Quartus Prime software, these settings are represented by assignments that map I/O settings, such as pin selection, and I/O standard and drive strength, to corresponding signals in your design.

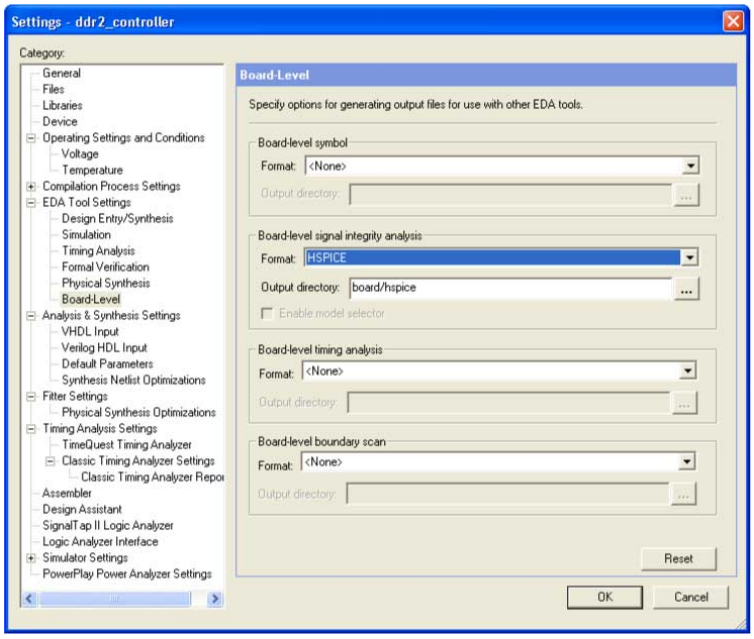
The Quartus Prime software provides multiple methods for creating these assignments:

- Using the Pin Planner
- Using the assignment editor
- Manually editing the **.qsf** file
- By making assignments in a scripted Quartus Prime flow using Tcl

14.5.4.2 Enabling HSPICE Writer

You must enable the HSPICE Writer in the **Settings** dialog box of the Quartus Prime software to generate the HSPICE decks from the Quartus Prime software.

Figure 101. EDA Tool Settings: Board Level Options Dialog Box



14.5.4.3 Enabling HSPICE Writer Using Assignments

You can also use HSPICE Writer in conjunction with a scripted Tcl flow. To enable HSPICE Writer during a full compile, include the following lines in your Tcl script.

Enable HSPICE Writer

```
set_global_assignment -name EDA_BOARD_DESIGN_SIGNAL_INTEGRITY_TOOL \
"HSPICE (Signal Integrity)"
set_global_assignment -name EDA_OUTPUT_DATA_FORMAT HSPICE \
-section_id eda_board_design_signal_integrity
set_global_assignment -name EDA_NETLIST_WRITER_OUTPUT_DIR <output_directory> \
-section_id eda_board_design_signal_integrity
```

As with command-line invocation, specifying the output directory is optional. If not specified, the output directory defaults to **board/hspice**.

14.5.4.4 Naming Conventions for HSPICE Files

HSPICE Writer automatically generates simulation files and names them using the following naming convention: *<device>_<pin #>_<pin_name>_<in/out>*.sp.

For bidirectional pins, two spice decks are produced; one with the I/O buffer configured as an input, and the other with the I/O buffer configured as an output.

The Quartus Prime software supports alphanumeric pin names that contain the underscore (_) and dash (-) characters. Any illegal characters used in file names are converted automatically to underscores.

Related Links

- Sample Output for I/O HSPICE Simulation Deck on page 252

A typical HSPICE simulation SPICE deck for an I/O-type output has several sections. Each section presents the simulation file one block at a time.

- Sample Input for I/O HSPICE Simulation Deck on page 248
  The following sections examine a typical HSPICE simulation spice deck for an I/O of type input. Each section presents the simulation file one block at a time.

14.5.4.5 Invoking HSPICE Writer

After HSPICE Writer is enabled, the HSPICE simulation files are generated automatically each time the project is completely compiled. The Quartus Prime software also provides an option to generate a new set of simulation files without having to recompile manually. In the Processing menu, click **Start EDA Netlist Writer** to generate new simulation files automatically.

Note: You must perform both Analysis & Synthesis and Fitting on a design before invoking the HSPICE Writer tool.

14.5.4.6 Invoking HSPICE Writer from the Command Line

If you use a script-based flow to compile your project, you can create HSPICE model files by including the following commands in your Tcl script (`.tcl` file).

Create HSPICE Model Files

```
set_global_assignment -name EDA_BOARD_DESIGN_SIGNAL_INTEGRITY_TOOL \
"HSPICE (Signal Integrity)"
set_global_assignment -name EDA_OUTPUT_DATA_FORMAT HSPICE \
-section_ideda_board_design_signal_integrity
set_global_assignment -name EDA_NETLIST_WRITER_OUTPUT_DIR <output_directory> \
-section_id eda_board_design_signal_integrity
```

The <*output_directory*> option specifies the location where HSPICE model files are saved. By default, the <*project directory*>**/board/hspice** directory is used.

Invoke HSPICE Writer

To invoke the HSPICE Writer tool through the command line, type:

```
quartus_eda.exe <project_name> --board_signal_integrity=on --format=HSPICE \
--output_directory=<output_directory>
```

<*output_directory*> specifies the location where the generated spice decks will be written (relative to the design directory). This is an optional parameter and defaults to **board/hspice**.

14.5.4.7 Customizing Automatically Generated HSPICE Decks

HSPICE models generated by the HSPICE Writer can be used for simulation as generated.

A default board description is included, and a default simulation is set up to measure rise and fall delays for both input and output simulations, which compensates for the double counting problem. However, Intel recommends that you customize the board description to more accurately represent your routing and termination scheme.

The sample board trace loading in the generated HSPICE model files must be replaced by your actual trace model before you can run a correct simulation. To do this, open the generated HSPICE model files for all pins you want to simulate and locate the following section.

Sample Board Trace Section

```
* I/O Board Trace and Termination Description
* - Replace this with your board trace and termination description
```

You must replace the example load with a load that matches the design of your PCB board. This includes a trace model, termination resistors, and, for output simulations, a receiver model. The spice circuit node that represents the pin of the FPGA package is called **pin**. The node that represents the far pin of the external device is called **load-in** (for output SPICE decks) and **source-in** (for input SPICE decks).

For an input simulation, you must also modify the stimulus portion of the spice file. The section of the file that must be modified is indicated in the following comment block.

Sample Source Stimulus Section

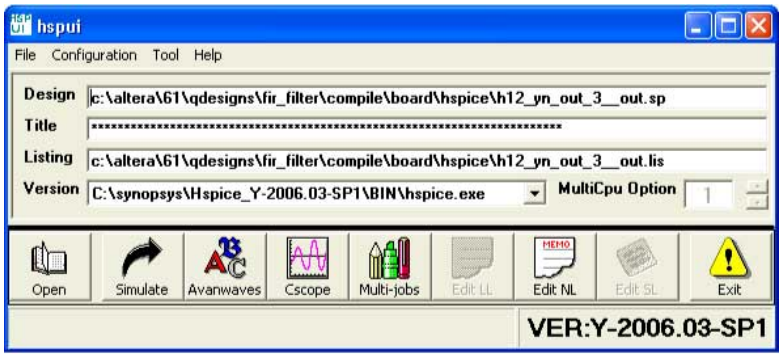
```
* Sample source stimulus placeholder
*  - Replace this with your I/O driver model
```

Replace the sample stimulus model with a model for the device that will drive the FPGA.

14.5.5 Running an HSPICE Simulation

Because simulation parameters are configured directly in the HSPICE model files, running a simulation requires only that you open an HSPICE file in the HSPICE user interface and start the simulation.

Figure 102. HSPICE User Interface Window



Click **Open** and browse to the location of the HSPICE model files generated by the Quartus Prime HSPICE Writer. The default location for HSPICE model files is <*project directory*>**/board/hspice**. Select the **.sp** file generated by the HSPICE Writer for the signal you want to simulate. Click **OK**.

To run the simulation, click **Simulate**. The status of the simulation is displayed in the window and saved in an **.lis** file with the same name as the .sp file when the simulation is complete. Check the **.lis** file if an error occurs during the simulation requiring a change in the .sp file to fix.

14.5.6 Interpreting the Results of an Output Simulation

By default, the automatically generated output simulation spice decks are set up to measure three delays for both rising and falling transitions. Two of the measurements, tpd_rise and tpd_fall, measure the double-counting corrected delay from the FPGA pin to the load pin. To determine the complete clock-edge to load-pin delay, add these numbers to the Quartus Prime software reported default loading t_{CO} delay.

The remaining four measurements, tpd_uncomp_rise, tpd_uncomp_fall, t_dblcnt_rise, and t_dblcnt_fall, are required for the double-counting compensation process and are not required for further timing usage.

Related Links

Simulation Analysis on page 252
The simulation analysis block of the simulation file is configured to measure the propagation delay from the source to the FPGA pin. Both the source and end point of the delay are referenced against the 50% V_{CCN} crossing point of the waveform.

14.5.7 Interpreting the Results of an Input Simulation

By default, the automatically generated input simulation SPICE decks are set up to measure delays from the source's driver pin to the FPGA's input pin for both rising and falling transitions.

The propagation delay is reported by HSPICE measure statements as tpd_rise and tpd_fall. To determine the complete source driver pin-to-FPGA register delay, add these numbers to the Quartus Prime software reported T_H and T_{SU} input timing numbers.

14.5.8 Viewing and Interpreting Tabular Simulation Results

The **.lis** file stores the collected simulation data in tabular form. The default simulation configured by the HSPICE Writer produces delay measurements for rising and falling transitions on both input and output simulations.

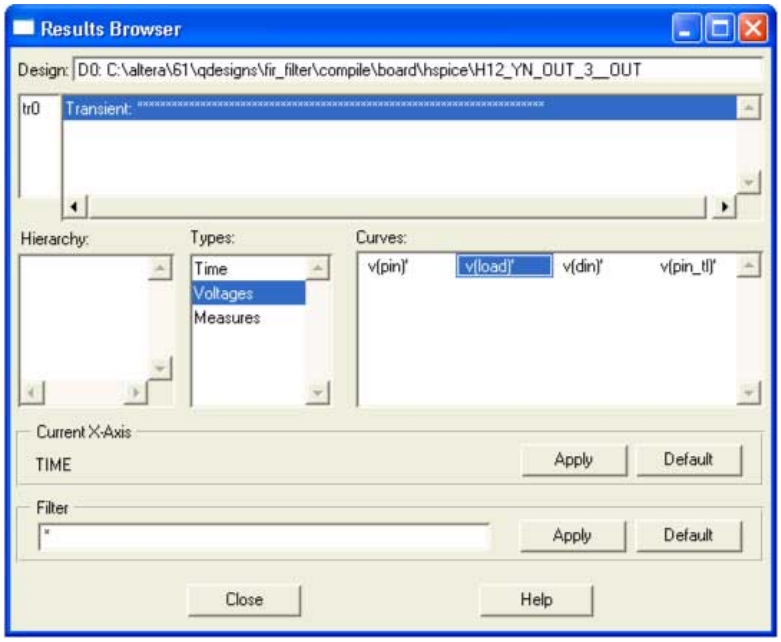
These measurements are found in the .lis file and named tpd_rise and tpd_fall. For output simulations, these values are already adjusted for the double count. To determine the complete delay from the FPGA logic to the load pin, add either of these measurements to the Quartus Prime t_{CO} delay. For input simulations, add either of these measurements to the Quartus Prime t_{SU} and t_H delay values to calculate the complete delay from the far end stimulus to the FPGA logic. Other values found in the **.lis** file, such as tpd_uncomp_rise, tpd_uncomp_fall, t_dblcnt_rise, and t_dblcnt_fall, are parts of the double count compensation calculation. These values are not necessary for further analysis.

14.5.9 Viewing Graphical Simulation Results

You can view the results of the simulation quickly as a graphical waveform display using the AvanWaves viewer included with HSPICE. With the default simulation configured by the HSPICE Writer, you can view the simulated waveforms at both the source and destination in input and output simulations.

To see the waveforms for the simulation, in the HSPICE user interface window, click **AvanWaves**. The AvanWaves viewer opens and displays the **Results Browser**.

Figure 103. HSPICE AvanWaves Results Browser



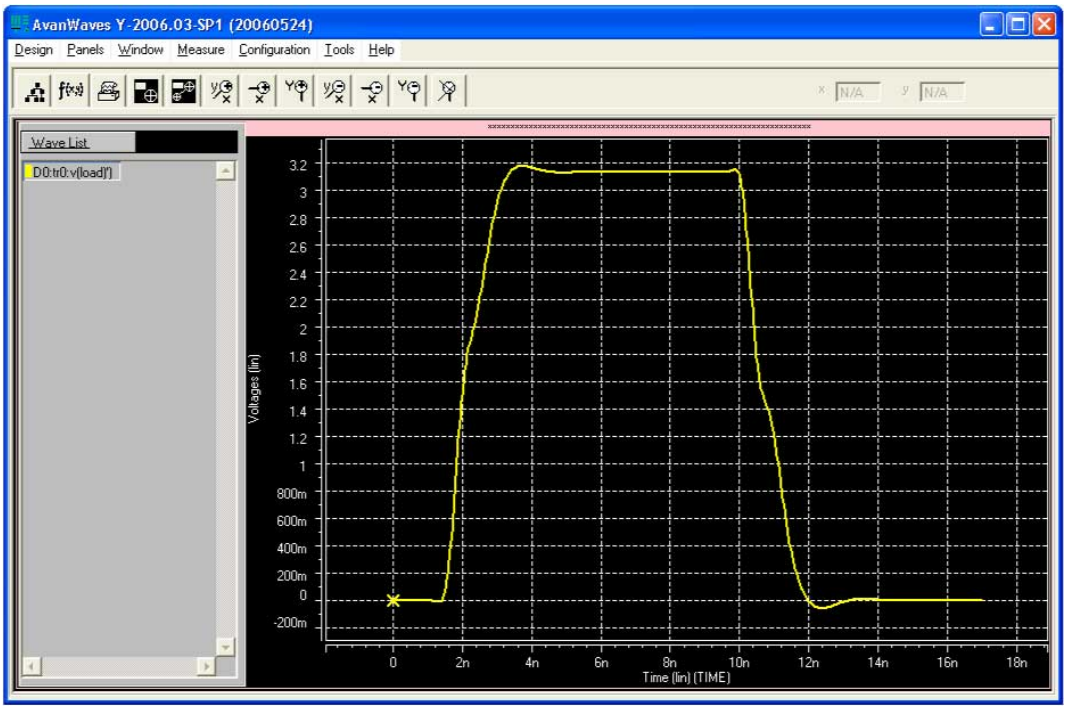
The **Results Browser** lets you select which waveform to view quickly in the main viewing window. If multiple simulations are run on the same signal, the list at the top of the **Results Browser** displays the results of each simulation. Click the simulation description to select which simulation to view. By default, the descriptions are derived from the first line of the HSPICE file, so the description might appear as a line of asterisks.

Select the type of waveform to view, by performing the following steps:

1. To see the source and destination waveforms with the default simulation, from the **Types** list, select **Voltages**.
2. On the **Curves** list, double-click the waveform you want to view. The waveform appears in the main viewing window.

You can zoom in and out and adjust the view as desired.

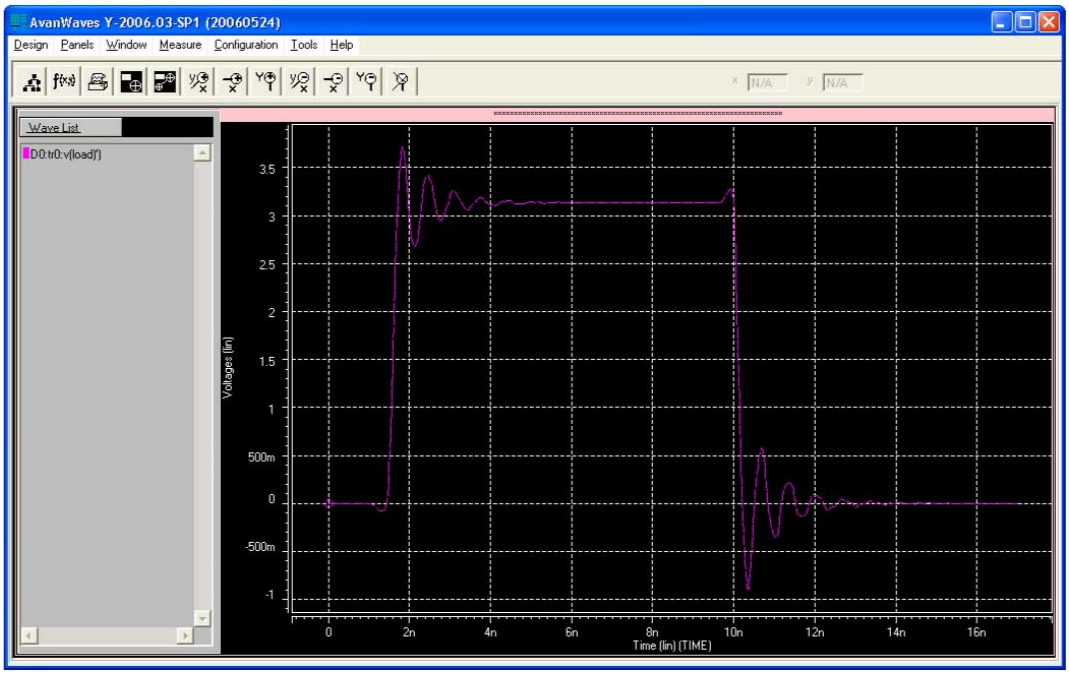
Figure 104. AvanWaves Waveform Viewer



14.5.10 Making Design Adjustments Based on HSPICE Simulations

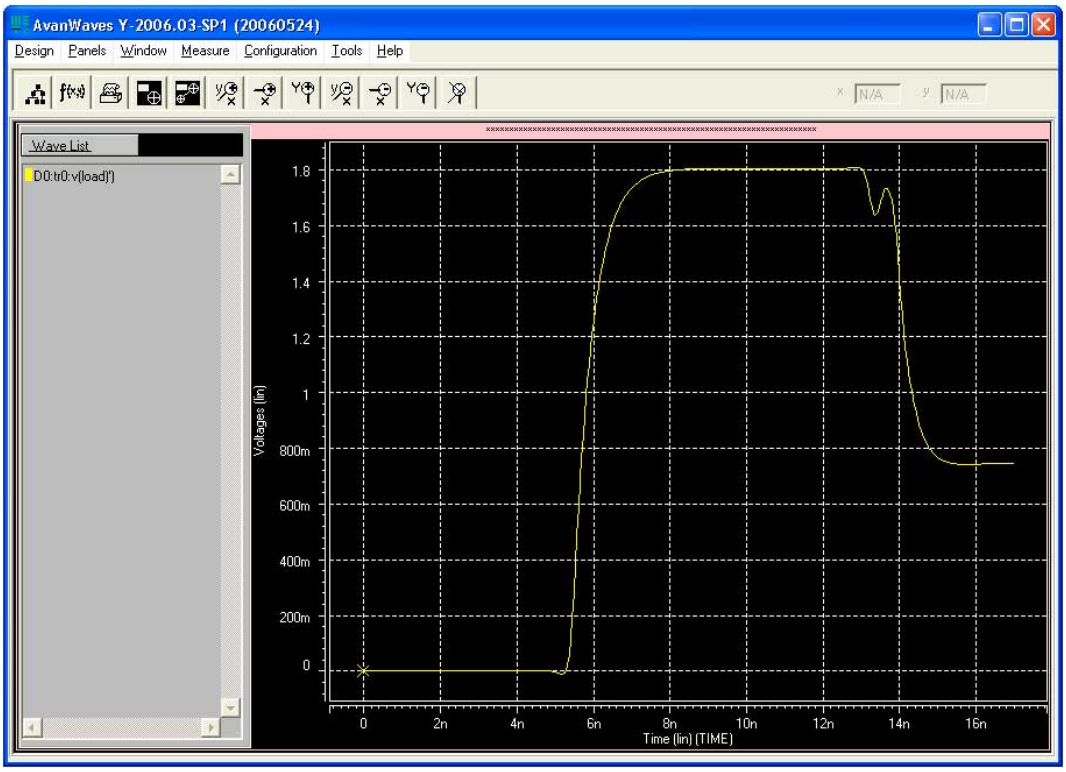
Based on the results of your simulations, you can make adjustments to the I/O assignments or simulation parameters if required. For example, after you run a simulation and see overshoot or ringing in the simulated signal at the destination buffer, you can adjust the drive strength I/O assignment setting to a lower value. Regenerate the HSPICE deck, and run the simulation again to verify that the change fixed the problem.

Figure 105. Example of Overshoot in the AvanWaves Waveform Viewer



If there is a discontinuity or any other anomalies at the destination, adjust the board description in the Quartus Prime Board Trace Model, or in the generated HSPICE model files to change the termination scheme or adjust termination component values. After making these changes, regenerate the HSPICE files if necessary, and rerun the simulation to verify whether your adjustments solved the problem.

Figure 106. Example of Signal Integrity Anomaly in the AvanWaves Waveform Viewer



Related Links

Altera Signal Integrity Center
For more information about board-level signal integrity and to learn about ways to improve it with simple changes to your FPGA design.

14.5.11 Sample Input for I/O HSPICE Simulation Deck

The following sections examine a typical HSPICE simulation spice deck for an I/O of type input. Each section presents the simulation file one block at a time.

14.5.11.1 Header Comment

The first block of an input simulation spice deck is the header comment. The purpose of this block is to provide an easily readable summary of how the simulation file has been automatically configured by the Quartus Prime software.

This block has two main components: The first component summarizes the I/O configuration relevant information such as device, speed grade, and so on. The second component specifies the exact test condition that the Quartus Prime software assumes for the given I/O standard.

Sample Header Comment Block

```
* Quartus Prime HSPICE Writer I/O Simulation Deck*

* This spice simulation deck was automatically generated by
```

```
* Quartus for the following IO settings:
*
*  Device:       EP2S60F1020C3
*  Speed Grade:  C3
*  Pin:          AA4 (out96)
*  Bank:         IO Bank 6 (Row I/O)
*  I/O Standard: LVTTL, 12mA
*  OCT:          Off
*
* Quartus Prime's default I/O timing delays assume the following slow
* corner simulation conditions.
*
*  Specified Test Conditions For Quartus Prime Tco
*    Temperature:      85C (Slowest Temperature Corner)
*    Transistor Model: TT (Typical Transistor Corner)
*    Vccn:             3.135V (Vccn_min = Nominal - 5%)
*    Vccpd:            2.97V (Vccpd_min = Nominal - 10%)
*    Load:             No Load
*    Vtt:              1.5675V (Voltage reference is Vccn/2)
*
* Note: The I/O transistors are specified to operate at least as
*       fast as the TT transistor corner, actual production
*       devices can be as fast as the FF corner. Any simulations
*       for hold times should be conducted using the fast process
*       corner with the following simulation conditions.
*          Temperature:      0C (Fastest Commercial Temperature Corner **)
*        Transistor Model: FF (Fastest Transistor Corner)
*        Vccn:             1.98V (Vccn_hold = Nominal + 10%)
*        Vccpd:            3.63V (Vccpd_hold = Nominal + 10%)
*        Vtt:              0.95V (Vtt_hold = Vccn/2 - 40mV)
*        Vcc:              1.25V (Vcc_hold = Maximum Recommended)
*        Package Model:    Short-circuit from pad to pin (no parasitics)
*
* Warnings:
```

14.5.11.2 Simulation Conditions

The simulation conditions block loads the appropriate process corner models for the transistors. This condition is automatically set up for the slow timing corner and is modified only if other simulation corners are desired.

Simulation Conditions Block

```
* Process Settings

.options brief
.inc 'sii_tt.inc' * TT process corner
```

14.5.11.3 Simulation Options

The simulation options block configures the simulation temperature and configures HSPICE with typical simulation options.

Simulation Options Block

```
* Simulation Options

.options brief=0
.options badchr co=132 scale=1e-6 acct ingold=2 nomod dv=1.0
+        dcstep=1 absv=1e-3 absi=1e-8 probe csdf=2 accurate=1
+        converge=1
.temp 85
```

Note: For a detailed description of these options, consult your *HSPICE* manual.

14.5.11.4 Constant Definition

The constant definition block of the simulation file instantiates the voltage sources that controls the configuration modes of the I/O buffer.

Constant Definition Block

```
* Constant Definition

voeb       oeb       0     vc  * Set to 0 to enable buffer output
vopdrain   opdrain   0     0   * Set to vc to enable open drain
vrambh     rambh     0     0   * Set to vc to enable bus hold
vrpullup   rpullup   0     0   * Set to vc to enable weak pullup
vpcdp5     rpcdp5    0     rp5 * Set the IO standard
vpcdp4     rpcdp4    0     rp4
vpcdp3     rpcdp3    0     rp3
vpcdp2     rpcdp2    0     rp2
vpcdp1     rpcdp1    0     rp1
vpcdp0     rpcdp0    0     rp0
vpcdn4     rpcdn4    0     rn4
vpcdn3     rpcdn3    0     rn3
vpcdn2     rpcdn2    0     rn2
vpcdn1     rpcdn1    0     rn1
vpcdn0     rpcdn0    0     rn0
vdin din       0     0
```

Where:

- Voltage source `voeb` controls the output enable of the buffer and is set to disabled for inputs.
- `vopdrain` controls the open drain mode for the I/O.
- `vrambh` controls the bus hold circuitry in the I/O.
- `vrpullup` controls the weak pullup.
- The next 11 voltages sources control the I/O standard of the buffer and are configured through a later library call.
- `vdin` is not used on input pins because it is the data pin for the output buffer.

14.5.11.5 Buffer Netlist

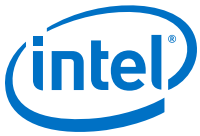
The buffer netlist block of the simulation spice deck loads all the load models required for the corresponding input pin.

Buffer Netlist Block

```
* IO Buffer Netlist

.include ‘vio_buffer.inc’
```

14.5.11.6 Drive Strength

The drive strength block of the simulation SPICE deck loads the configuration bits necessary to configure the I/O into the proper I/O standard and drive strengths.



Although these settings are not relevant to an input buffer, they are provided to allow the SPICE deck to be modifiable to support bidirectional simulations.

Drive Strength Block

```
* Drive Strength Settings

.lib 'drive_select_hio.lib' 3p3ttl_12ma
```

14.5.11.7 I/O Buffer Instantiation

The I/O buffer instantiation block of the simulation SPICE deck instantiates the necessary power supplies and I/O model components that are necessary to simulate the given I/O.

I/O Buffer Instantiation

```
I/O Buffer Instantiation

* Supply Voltages Settings
.param vcn=3.135
.param vpd=2.97
.param vc=1.15

* Instantiate Power Supplies|
vvcc       vcc       0     vc     * FPGA core voltage
vvss       vss       0     0      * FPGA core ground
vvccn      vccn      0     vcn    * IO supply voltage
vvssn      vssn      0     0      * IO ground
vvccpd     vccpd     0     vpd    * Pre-drive supply voltage

* Instantiate I/O Buffer
xvio_buf din oeb opdrain die rambh
+ rpcdn4 rpcdn3 rpcdn2 rpcdn1 rpcdn0
+ rpcdp5 rpcdp4 rpcdp3 rpcdp2 rpcdp1 rpcdp0
+ rpullup vccn vccpd vcpad0 vio_buf

* Internal Loading on Pad
* - No loading on this pad due to differential buffer/support
*   circuitry

* I/O Buffer Package Model
* - Single-ended I/O standard on a Row I/O
.lib 'lib/package.lib' hio
xpkg die pin hio_pkg
```

14.5.11.8 Board Trace and Termination

The board trace and termination block of the simulation SPICE deck is provided only as an example. Replace this block with your own board trace and termination models.

Board Trace and Termination Block

```
* I/O Board Trace and Termination Description
* - Replace this with your board trace and termination description

wtline pin vssn load vssn N=1 L=1 RLGCMODEL=tlinemodel
.MODEL tlinemodel W MODELTYPE=RLGC N=1 Lo=7.13n Co=2.85p
Rterm2 load vssn 1x
```

14.5.11.9 Stimulus Model

The stimulus model block of the simulation spice deck is provided only as a place holder example. Replace this block with your own stimulus model. Options for this include an IBIS or HSPICE model, among others.

Stimulus Model Block

```
* Sample source stimulus placeholder
* - Replace this with your I/O driver model

Vsource source 0 pulse(0 vcn 0s 0.4ns 0.4ns 8.5ns 17.4ns)
```

14.5.11.10 Simulation Analysis

The simulation analysis block of the simulation file is configured to measure the propagation delay from the source to the FPGA pin. Both the source and end point of the delay are referenced against the 50% V_{CCN} crossing point of the waveform.

Simulation Analysis Block

```
* Simulation Analysis Setup

* Print out the voltage waveform at both the source and the pin
.print tran v(source)  v(pin)
.tran 0.020ns 17ns

* Measure the propagation delay from the source pin to the pin
* referenced against the 50% voltage threshold crossing point

.measure TRAN tpd_rise TRIG v(source) val='vcn*0.5' rise=1
+ TARG v(pin) val ='vcn*0.5' rise=1
.measure TRAN tpd_fall TRIG v(source) val='vcn*0.5' fall=1
+ TARG v(pin) val ='vcn*0.5' fall=1
```

14.5.12 Sample Output for I/O HSPICE Simulation Deck

A typical HSPICE simulation SPICE deck for an I/O-type output has several sections. Each section presents the simulation file one block at a time.

14.5.12.1 Header Comment

The first block of an output simulation SPICE deck is the header comment. The purpose of this block is to provide a readable summary of how the simulation file has been automatically configured by the Quartus Prime software.

This block has two main components:

- The first component summarizes the I/O configuration relevant information such as device, speed grade, and so on.
- The second component specifies the exact test condition that the Quartus Prime software assumes when generating t_{CO} delay numbers. This information is used as part of the double-counting correction circuitry contained in the simulation file.

The SPICE decks are preconfigured to calculate the slow process corner delay but can also be used to simulate the fast process corner as well. The fast corner conditions are listed in the header under the notes section.

The final section of the header comment lists any warning messages that you must consider when you use the SPICE decks.

Header Comment Block

```
* Quartus Prime HSPICE Writer I/O Simulation Deck
*
* This spice simulation deck was automatically generated by
* Quartus Prime for the following IO settings:
*
*  Device:       EP2S60F1020C3
*  Speed Grade:  C3
*  Pin:          AA4 (out96)
*  Bank:         IO Bank 6 (Row I/O)
*    I/O Standard: LVTTL, 12mA
*  OCT:          Off
*
* Quartus' default I/O timing delays assume the following slow
* corner simulation conditions.
*  Specified Test Conditions For Quartus Prime Tco
*    Temperature:      85C (Slowest Temperature Corner)
*    Transistor Model: TT (Typical Transistor Corner)
*    Vccn:             3.135V (Vccn_min = Nominal - 5%)
*    Vccpd:            2.97V (Vccpd_min = Nominal - 10%)
*    Load:             No Load
*    Vtt:              1.5675V (Voltage reference is Vccn/2)
* For C3 devices, the TT transistor corner provides an
* approximation for worst case timing. However, for functionality
* simulations, it is recommended that the SS corner be simulated
* as well.
*
* Note: The I/O transistors are specified to operate at least as
*       fast as the TT transistor corner, actual production
*       devices can be as fast as the FF corner. Any simulations
*       for hold times should be conducted using the fast process
*       corner with the following simulation conditions.
*             Temperature:      0C (Fastest Commercial Temperature Corner **)
*             Transistor Model: FF (Fastest Transistor Corner)
*             Vccn:             1.98V (Vccn_hold = Nominal + 10%)
*             Vccpd:            3.63V (Vccpd_hold = Nominal + 10%)
*         Vtt:              0.95V (Vtt_hold = Vccn/2 - 40mV)
*         Vcc:              1.25V (Vcc_hold = Maximum Recommended)
*         Package Model:    Short-circuit from pad to pin
* Warnings:
```

14.5.12.2 Simulation Conditions

The simulation conditions block loads the appropriate process corner models for the transistors. This condition is automatically set up for the slow timing corner and must be modified only if other simulation corners are desired.

Simulation Conditions Block

```
* Process Settings

.options brief
.inc 'sii_tt.inc' * typical-typical process corner
```

Note: Two separate corners cannot be simulated at the same time. Instead, simulate the base case using the Quartus corner as one simulation and then perform a second simulation using the desired customer corner. The results of the two simulations can be manually added together.

14.5.12.3 Simulation Options

The simulation options block configures the simulation temperature and configures HSPICE with typical simulation options.

Simulation Options Block

```
* Simulation Options
.options brief=0
.options badchr co=132 scale=1e-6 acct ingold=2 nomod dv=1.0
+        dcstep=1 absv=1e-3 absi=1e-8 probe csdf=2 accurate=1
+        converge=1
.temp 85
```

Note: For a detailed description of these options, consult your *HSPICE* manual.

14.5.12.4 Constant Definition

The constant definition block of the output simulation SPICE deck instantiates the voltage sources that controls the configuration modes of the I/O buffer.

Constant Definition Block

```
* Constant Definition

voeb       oeb       0     0 * Set to 0 to enable buffer output
vopdrain   opdrain   0     0 * Set to vc to enable open drain
vrambh     rambh     0     0 * Set to vc to enable bus hold
vrpullup   rpullup   0     0 * Set to vc to enable weak pullup
vpci       rpci      0     0 * Set to vc to enable pci mode
vpcdp4     rpcdp4    0     rp4  * These control bits set the IO standard
vpcdp3     rpcdp3    0     rp3
vpcdp2     rpcdp2    0     rp2
vpcdp1     rpcdp1    0     rp1
vpcdp0     rpcdp0    0     rp0
vpcdn4     rpcdn4    0     rn4
vpcdn3     rpcdn3    0     rn3
vpcdn2     rpcdn2    0     rn2
vpcdn1     rpcdn1    0     rn1
vpcdn0     rpcdn0    0     rn0
vdin       din       0     pulse(0 vc 0s 0.2ns 0.2ns 8.5ns 17.4ns)
```

Where:

- Voltage source `voeb` controls the output enable of the buffer.
- `vopdrain` controls the open drain mode for the I/O.
- `vrambh` controls the bus hold circuitry in the I/O.
- `vrpullup` controls the weak pullup.
- `vpci` controls the PCI clamp.
- The next ten voltage sources control the I/O standard of the buffer and are configured through a later library call.
- `vdin` is connected to the data input of the I/O buffer.
- The edge rate of the input stimulus is automatically set to the correct value by the Quartus Prime software.

14.5.12.5 I/O Buffer Netlist

The I/O buffer netlist block loads all of the models required for the corresponding pin. These include a model for the I/O output buffer, as well as any loads that might be present on the pin.

I/O Buffer Netlist Block

```
*IO Buffer Netlist

.include ‘hio_buffer.inc’
.include ‘lvds_input_load.inc’
.include ‘lvds_oct_load.inc’
```

14.5.12.6 Drive Strength

The drive strength block of the simulation spice deck loads the configuration bits for configuring the I/O to the proper I/O standard and drive strength. These options are set by the HSPICE Writer tool and are not changed for expected use.

Drive Strength Block

```
* Drive Strength Settings

.lib ‘drive_select_hio.lib’ 3p3ttl_12ma
```

14.5.12.7 Slew Rate and Delay Chain

Stratix and Cyclone devices have sections for configuring the slew rate and delay chain settings.

Slew Rate and Delay Chain Settings

```
* Programmable Output Delay Control Settings

.lib ‘lib/output_delay_control.lib’ no_delay

* Programmable Slew Rate Control Settings

.lib ‘lib/slew_rate_control.lib’ slow_slow
```

14.5.12.8 I/O Buffer Instantiation

The I/O buffer instantiation block of the output simulation spice deck instantiates the necessary power supplies and I/O model components that are necessary to simulate the given I/O.

I/O Buffer Instantiation Block

```
* I/O Buffer Instantiation

* Supply Voltages Settings
.param vcn=3.135
.param vpd=2.97
.param vc=1.15

* Instantiate Power Supplies
vvcc       vcc       0     vc     * FPGA core voltage
```

```
vvss       vss       0     0      * FPGA core ground
vvccn      vccn      0     vcn    * IO supply voltage
vvssn      vssn      0     0      * IO ground
vvccpd     vccpd     0     vpd    * Pre-drive supply voltage

* Instantiate I/O Buffer
xhio_buf din oeb opdrain die rambh
+ rpcdn4 rpcdn3 rpcdn2 rpcdn1 rpcdn0
+ rpcdp4 rpcdp3 rpcdp2 rpcdp1 rpcdp0
+ rpullup vccn vccpd vcpad0 hio_buf

* Internal Loading on Pad
* - This pad has an LVDS input buffer connected to it, along
*   with differential OCT circuitry. Both are disabled but
*   introduce loading on the pad that is modeled below.
xlvds_input_load die vss vccn lvds_input_load
xlvds_oct_load die vss vccpd vccn vcpad0 vccn lvds_oct_load

* I/O Buffer Package Model
* - Single-ended I/O standard on a Row I/O
.lib 'lib/package.lib' hio
xpkg die pin hio_pkg
```

14.5.12.9 Board and Trace Termination

The board trace and termination block of the simulation SPICE deck is provided only as an example. Replace this block with your specific board loading models.

Board Trace and Termination Block

```
* I/O Board Trace And Termination Description
* - Replace this with your board trace and termination description
wtline pin vssn load vssn N=1 L=1 RLGCMODEL=tlinemodel
.MODEL tlinemodel W MODELTYPE=RLGC N=1 Lo=7.13n Co=2.85p
Rterm2 load vssn 1x
```

14.5.12.10 Double-Counting Compensation Circuitry

The double-counting compensation circuitry block of the simulation SPICE deck instantiates a second I/O buffer that is used to measure double-counting. The buffer is configured identically to the user I/O buffer but is connected to the Quartus Prime software test load. The simulated delay of this second buffer can be interpreted as the amount of double-counting between the Quartus Prime software and HSPICE Writer simulated results.

As the amount of double-counting is constant for a given I/O standard on a given pin, consider separating the double-counting circuitry from the simulation file. In doing so, you can perform any number of I/O simulations while referencing the delay only once.

(Part of)Double-Counting Compensation Circuitry Block

```
* Double Counting Compensation Circuitry
*
* The following circuit is designed to calculate the amount of
* double counting between Quartus Prime and the HSPICE models. If
* you have not changed the default simulation temperature or
* transistor corner the double counting will be automatically
* compensated by this spice deck. In the event you wish to
* simulate an IO at a different temperature or transistor corner
* you will need to remove this section of code and manually
* account for double counting. A description of Intel's
* recommended procedure for this process can be found in the
```

```
* Quartus Prime HSPICE Writer AppNote.
*

* Supply Voltages Settings
.param vcn_tl=3.135
.param vpd_tl=2.97

* Test Load Constant Definition
vopdrain_tl   opdrain_tl   0     0
vrambh_tl     rambh_tl     0     0
vrpullup_tl   rpullup_tl   0     0

* Instantiate Power Supplies
vvccn_tl      vccn_tl      0     vcn_tl
vvssn_tl      vssn_tl      0     0
vvccpd_tl     vccpd_tl     0     vpd_tl

* Instantiate I/O Buffer
xhio_testload din oeb opdrain_tl die_tl rambh_tl
+ rpcdn4 rpcdn3 rpcdn2 rpcdn1 rpcdn0
+ rpcdp4 rpcdp3 rpcdp2 rpcdp1 rpcdp0
+ rpullup_tl vccn_tl vccpd_tl vcpad0_tl hio_buf

* Internal Loading on Pad
xlvds_input_testload die_tl vss vccn_tl lvds_input_load
xlvds_oct_testload die_tl vss vccpd_tl vccn_tl vcpad0_tl vccn_tl
lvds_oct_load

* I/O Buffer Package Model
* - Single-ended I/O standard on a Row I/O
.lib 'lib/package.lib' hio
xpkg die pin hio_pkg

* Default Intel Test Load
* - 3.3V LVTTL default test condition is an open load
```

Related Links

The Double Counting Problem in HSPICE Simulations on page 238
Simulating I/Os using accurate models is extremely helpful for finding and fixing FPGA I/O timing and board signal integrity issues before any boards are built. However, the usefulness of such simulations is directly related to the accuracy of the models used and whether the simulations are set up and performed correctly. To ensure accuracy in models and simulations created for FPGA output signals, the timing hand-off between t_{CO} timing in the Quartus Prime software and simulation-based board delay must be taken into account. If this hand-off is not handled correctly, the calculated delay could either count some of the delay twice or even miss counting some of the delay entirely.

14.5.12.11 Simulation Analysis

The simulation analysis block is set up to measure double-counting corrected delays. This is accomplished by measuring the uncompensated delay of the I/O buffer when connected to the user load, and when subtracting the simulated amount of double-counting from the test load I/O buffer.

Simulation Analysis Block

```
*Simulation Analysis Setup

* Print out the voltage waveform at both the pin and far end load
.print tran v(pin) v(load)
.tran 0.020ns 17ns
```

```
* Measure the propagation delay to the load pin. This value will
* include some double counting with Quartus Prime's Tco
.measure TRAN tpd_uncomp_rise TRIG v(din) val='vc*0.5' rise=1
+ TARG v(load) val='vcn*0.5' rise=1
.measure TRAN tpd_uncomp_fall TRIG v(din) val='vc*0.5' fall=1
+ TARG v(load) val='vcn*0.5' fall=1

* The test load buffer can calculate the amount of double counting
.measure TRAN t_dblcnt_rise TRIG v(din) val='vc*0.5' rise=1
+ TARG v(pin_tl) val='vcn_tl*0.5' rise=1
.measure TRAN t_dblcnt_fall TRIG v(din) val='vc*0.5' fall=1
+ TARG v(pin_tl) val='vcn_tl*0.5' fall=1

* Calculate the true propagation delay by subtraction
.measure TRAN tpd_rise PARAM='tpd_uncomp_rise-t_dblcnt_rise'
.measure TRAN tpd_fall PARAM='tpd_uncomp_fall-t_dblcnt_fall'
```

14.5.13 Advanced Topics

The information in this section describes some of the more advanced topics and methods employed when setting up and running HSPICE simulation files.

14.5.13.1 PVT Simulations

The automatically generated HSPICE simulation files are set up to simulate the slow process corner using low voltage, high temperature, and slow transistors. To ensure a fully robust link, Intel recommends that you run simulations over all process corners.

To perform process, voltage, and temperature (PVT) simulations, manually modify the spice decks in a two step process:

1. Remove the double-counting compensation circuitry from the simulation file. This is required as the amount of double-counting is dependant upon how the Quartus Prime software calculates delays and is not based on which PVT corner is being simulated. By default, the Quartus Prime software provides timing numbers using the slow process corner.
2. Select the proper corner for the PVT simulation by setting the correct HSPICE temperature, changing the supply voltage sources, and loading the correct transistor models.

A more detailed description of HSPICE process corners can be found in the family-specific HSPICE model documentation.

Related Links

Accessing HSPICE Simulation Kits on page 237
You can access the available HSPICE models with the Quartus Prime software's HSPICE Writer tool and also at the Spice Models for Intel Devices web page.

14.5.13.2 Hold Time Analysis

Intel recommends performing worst-case hold time analysis using the fast corner models, which use fast transistors, high voltage, and low temperature. This involves modifying the SPICE decks to select the correct temperature option, change the supply voltage sources, and load the correct fast transistor models. The values of these parameters are located in the header comment section of the corresponding simulation deck files.

For a truly worst-case analysis, combine the HSPICE Writer hold time analysis results with the Quartus Prime software fast timing model. This requires that you change the double-counting compensation circuitry in the simulations files to also simulate the fast process corners, as this is what the Quartus Prime software uses for the fast timing model.

Note: This method of hold time analysis is recommended only for globally synchronous buses. Do not apply this method of hold-time analysis to source synchronous buses. This is because the source synchronous clocking scheme is designed to cancel out some of the PVT timing effects. If this is not taken into account, the timing results will not be accurate. Proper source synchronous timing analysis is beyond the scope of this document.

14.5.13.3 I/O Voltage Variations

Use each of the FPGA family datasheets to verify the recommended operating conditions for supply voltages. For current FPGA families, the maximum recommended voltage corresponds to the fast corner, while the minimum recommended voltage corresponds to the slow corner. These voltage recommendations are specified at the power pins of the FPGA and are not necessarily the same voltage that are seen by the I/O buffers due to package IR drops.

The automatically generated HSPICE simulation files model this IR effect pessimistically by including a 50-mV IR drop on the V_{CCPD} supply when a high drive strength standard is being used.

14.5.13.4 Correlation Report

Correlation reports for the HSPICE I/O models are located in the family-specific HSPICE I/O buffer simulation kits.

Related Links

Accessing HSPICE Simulation Kits on page 237
You can access the available HSPICE models with the Quartus Prime software's HSPICE Writer tool and also at the Spice Models for Intel Devices web page.

14.6 Document Revision History

Table 60. Document Revision History

Date	Version	Changes
2016.10.31	16.1.0	• Implemented Intel rebranding. • Corrected statement about timing simulation and double counting.
2015.11.02	15.1.0	• Changed instances of *Quartus II* to *Quartus Prime*.
June 2014	14.0.0	Updated format.
December 2010	10.0.1	Template update.
July 2010	10.0.0	Updated device support.
November 2009	9.1.0	No change to content.

continued...

Date	Version	Changes
March 2009	9.0.0	• Was volume 3, chapter 12 in the 8.1.0 release. • No change to content.
November 2008	8.1.0	• Changed to 8-1/2 x 11 page size. • Added information for Stratix III devices. • Input signals for Cyclone III devices are supported.
May 2008	8.0.0	• Updated "Introduction" on page 12–1. • Updated Figure 12–1. • Updated Figure 12–3. • Updated Figure 12–13. • Updated "Output File Generation" on page 12–6. • Updated "Simulation with HSPICE Models" on page 12–17. • Updated "Invoking HSPICE Writer from the Command Line" on page 12–22. • Added "Sample Input for I/O HSPICE Simulation Deck" on page 12–29. • Added "Sample Output for I/O HSPICE Simulation Deck" on page 12–33. • Updated "Correlation Report" on page 12–41. • Added hyperlinks to referenced documents and websites throughout the chapter. • Made minor editorial updates.

Related Links

Altera Documentation Archive

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

15 Cadence PCB Design Tools Support

15.1 Cadence PCB Design Tools Support

The Quartus Prime software interacts with the following software to provide a complete FPGA-to-board integration design workflow: the Cadence Allegro Design Entry HDL software and the Cadence Allegro Design Entry CIS (Component Information System) software (also known as OrCAD Capture CIS). The information is useful for board design and layout engineers who want to begin the FPGA board integration process while the FPGA is still in the design phase. Part librarians can also benefit by learning the method to use output from the Quartus Prime software to create new library parts and symbols.

With today's large, high-pin-count and high-speed FPGA devices, good PCB design practices are important to ensure the correct operation of your system. The PCB design takes place concurrently with the design and programming of the FPGA. An FPGA or ASIC designer initially creates the signal and pin assignments and the board designer must transfer these assignments to the symbols used in their system circuit schematics and board layout correctly. As the board design progresses, you must perform pin reassignments to optimize the layout. You must communicate pin reassignments to the FPGA designer to ensure the new assignments are processed through the FPGA with updated placement and routing.

You require the following software:

- The Quartus Prime software version 5.1 or later
- The Cadence Allegro Design Entry HDL software or the Cadence Allegro Design Entry CIS software version 15.2 or later
- The OrCAD Capture software with the optional CIS option version 10.3 or later (optional)

Note: These programs are very similar because the Cadence Allegro Design Entry CIS software is based on the OrCAD Capture software. Any procedural information can also apply to the OrCAD Capture software unless otherwise noted.

Related Links

- www.cadence.com
  For more information about obtaining and licensing the Cadence tools and for product information, support, and training
- www.orcad.com
  For more information about the OrCAD Capture software and the CIS option
- www.ema-eda.com
  For more information about Cadence and OrCAD support and training.

15.2 Product Comparison

Table 61. Cadence and OrCAD Product Comparison

Description	Cadence Allegro Design Entry HDL	Cadence Allegro Design Entry CIS	OrCAD Capture CIS
Former Name	Concept HDL Expert	Capture CIS Studio	—
History	More commonly known by its former name, Cadence renamed all board design tools in 2004 under the Allegro name.	Based directly on OrCAD Capture CIS, the Cadence Allegro Design Entry CIS software is still developed by OrCAD but sold and marketed by Cadence. EMA provides support and training.	The basis for Design Entry CIS is still developed by OrCAD for continued use by existing OrCAD customers. EMA provides support and training for all OrCAD products.
Vendor Design Flow	Cadence Allegro 600 series, formerly known as the Expert Series, for high-end, high-speed design.	Cadence Allegro 200 series, formerly known as the Studio Series, for small- to medium-level design.	—

Related Links

- www.cadence.com
- www.ema-eda.com

15.3 FPGA-to-PCB Design Flow

You can create a design flow integrating an Intel FPGA design from the Quartus Prime software through a circuit schematic in the Cadence Allegro Design Entry HDL software or the Cadence Allegro Design Entry CIS software.

Figure 107. Design Flow with the Cadence Allegro Design Entry HDL Software

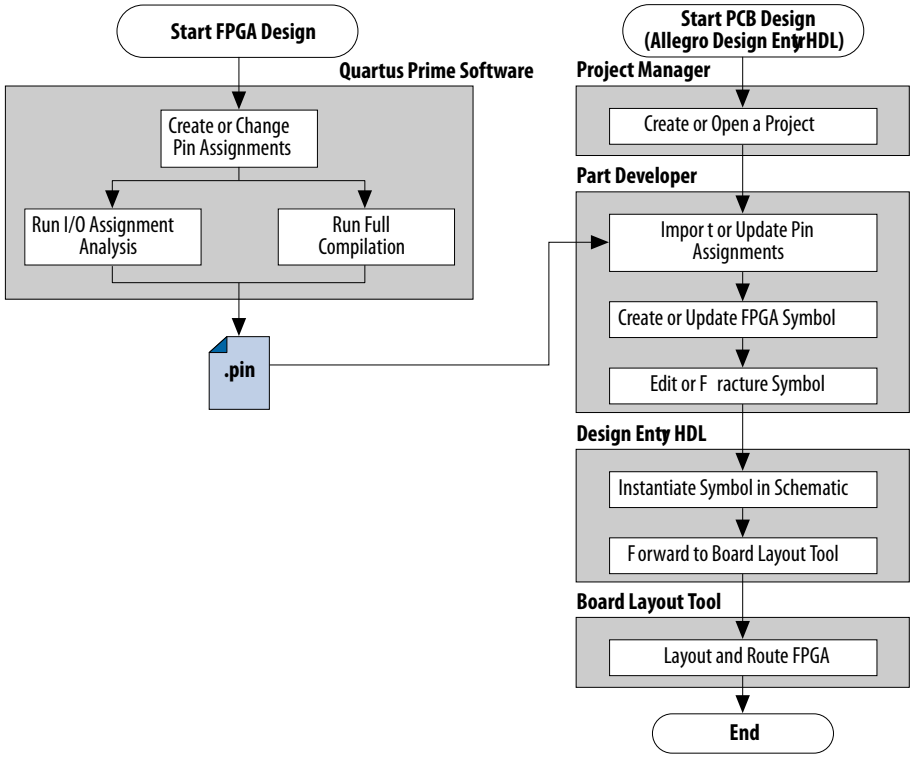
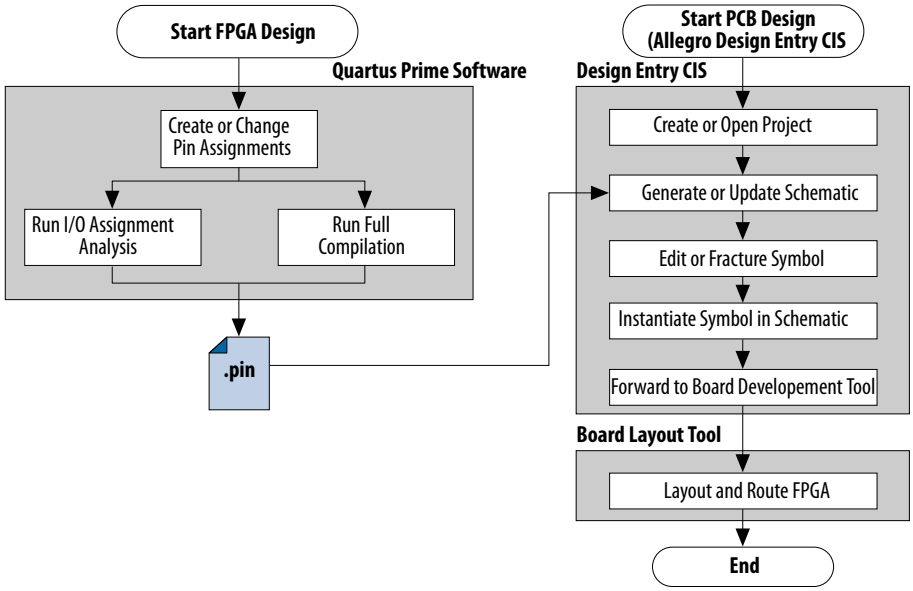


Figure 108. Design Flow with the Cadence Allegro Design Entry CIS Software



To create FPGA symbols using the Cadence Allegro PCB Librarian Part Developer tool, you must obtain the Cadence PCB Librarian Expert license. You can update symbols with changes made to the FPGA design using any of these tools.

15.3.1 Integrating Intel FPGA Design

To integrate an Intel FPGA design starting in the Quartus Prime software through to a circuit schematic in the Cadence Allegro Design Entry HDL software or the Cadence Allegro Design Entry CIS software, follow these steps:

1. In the Quartus Prime software, compile your design to generate a Pin-Out File (**.pin**) to transfer the assignments to the Cadence software.
2. If you are using the Cadence Allegro Design Entry HDL software for your schematic design, follow these steps:
   a. Open an existing project or create a new project in the Cadence Allegro Project Manager tool.
   b. Construct a new symbol or update an existing symbol using the Cadence Allegro PCB Librarian Part Developer tool.
   c. With the Cadence Allegro PCB Librarian Part Developer tool, edit your symbol or fracture it into smaller parts (optional).
   d. Instantiate the symbol in your Cadence Allegro Design Entry HDL software schematic and transfer the design to your board layout tool.

      or

      If you are using the Cadence Allegro Design Entry CIS software for your schematic design, follow these steps:
   e. Generate a new part in a new or existing Cadence Allegro Design Entry CIS project, referencing the **.pin** output file from the Quartus Prime software. You can also update an existing symbol with a new **.pin**.
   f. Split the symbol into smaller parts as necessary.
   g. Instantiate the symbol in your Cadence Allegro Design Entry CIS schematic and transfer the design to your board layout tool.

15.4 Setting Up the Quartus Prime Software

You can transfer pin and signal assignments from the Quartus Prime software to the Cadence design tools by generating the Quartus Prime project **.pin**. The **.pin** is an output file generated by the Quartus Prime Fitter containing pin assignment information. You can use the Quartus Prime Pin Planner to set and change the assignments in the **.pin** and then transfer the assignments to the Cadence design tools. You cannot, however, import pin assignment changes from the Cadence design tools into the Quartus Prime software with the **.pin**.

The **.pin** lists all used and unused pins on your selected Intel device. The **.pin** also provides the following basic information fields for each assigned pin on the device:

- Pin signal name and usage
- Pin number
- Signal direction
- I/O standard

- Voltage
- I/O bank
- User or Fitter-assigned

Related Links

I/O Management on page 18
For more information about using the Quartus Prime Pin Planner to create or change pin assignment details.

15.4.1 Generating a .pin File

To generate a **.pin**, follow these steps:

1. Compile your design.
2. Locate the **.pin** in your Quartus Prime project directory with the name <project name>**.pin**.

Related Links

I/O Management on page 18
For more information about pin and signal assignment transfer and the files that the Quartus Prime software can import and export.

15.5 FPGA-to-Board Integration with the Cadence Allegro Design Entry HDL Software

The Cadence Allegro Design Entry HDL software is a schematic capture tool and is part of the Cadence 600 series design flow. Use the Cadence Allegro Design Entry HDL software to create flat circuit schematics for all types of PCB design. The Cadence Allegro Design Entry HDL software can also create hierarchical schematics to facilitate design reuse and team-based design. With the Cadence Allegro Design Entry HDL software, the design flow from FPGA-to-board is one-way, using only the **.pin** generated by the Quartus Prime software. You can only make signal and pin assignment changes in the Quartus Prime software and these changes reflect as updated symbols in a Cadence Allegro Design Entry HDL project.

For more information about the design flow with the Cadence Allegro Design Entry HDL software, refer to Figure 107 on page 263.

Note: Routing or pin assignment changes made in a board layout tool or a Cadence Allegro Design Entry HDL software symbol cannot be back-annotated to the Quartus Prime software.

Related Links

www.cadence.com
Provides information about the Cadence Allegro Design Entry HDL software and the Cadence Allegro PCB Librarian Part Developer tool, including licensing, support, usage, training, and product updates.

15.5.1 Creating Symbols

In addition to circuit simulation, circuit board schematic creation is one of the first tasks required when designing a new PCB. Schematics must understand how the PCB works, and to generate a netlist for a board layout tool for board design and routing. The Cadence Allegro PCB Librarian Part Developer tool allows you to create schematic symbols based on FPGA designs exported from the Quartus Prime software.

You can create symbols for the Cadence Allegro Design Entry HDL project with the Cadence Allegro PCB Librarian Part Developer tool, which is available in the Cadence Allegro Project Manager tool. Intel recommends using the Cadence Allegro PCB Librarian Part Developer tool to import FPGA designs into the Cadence Allegro Design Entry HDL software.

You must obtain a PCB Librarian Expert license from Cadence to run the Cadence Allegro PCB Librarian Part Developer tool. The Cadence Allegro PCB Librarian Part Developer tool provides a GUI with many options for creating, editing, fracturing, and updating symbols. If you do not use the Cadence Allegro PCB Librarian Part Developer tool, you must create and edit symbols manually in the Symbol Schematic View in the Cadence Allegro Design Entry HDL software.

Note: If you do not have a PCB Librarian Expert license, you can automatically create FPGA symbols using the programmable IC (PIC) design flow found in the Cadence Allegro Project Manager tool.

Before creating a symbol from an FPGA design, you must open a Cadence Allegro Design Entry HDL project with the Cadence Allegro Project Manager tool. If you do not have an existing Cadence Allegro Design Entry HDL project, you can create one with the Cadence Allegro Design Entry HDL software. The Cadence Allegro Design Entry HDL project directory with the name <project name>`.cpm` contains your Cadence Allegro Design Entry HDL projects.

While the Cadence Allegro PCB Librarian Part Developer tool refers to symbol fractures as slots, the other tools use different names to refer to symbol fractures.

Table 62. Symbol Fracture Naming Conventions

	Cadence Allegro PCB Librarian Part Developer Tool	Cadence Allegro Design Entry HDL Software	Cadence Allegro Design Entry CIS Software
During symbol generation	Slots	—	Sections
During symbol schematic instantiation	—	Versions	Parts

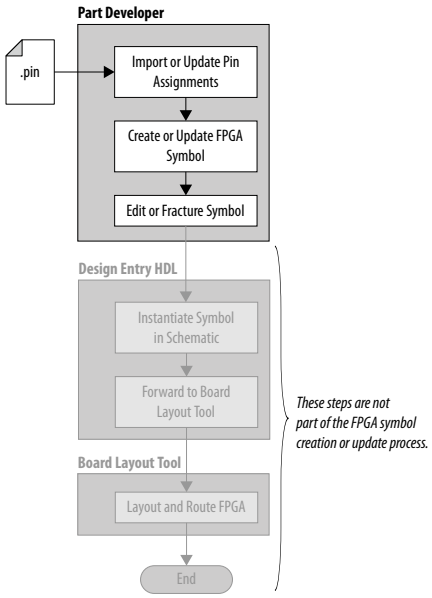
Related Links

www.cadence.com
Provides information about using the PIC design flow.

15.5.1.1 Cadence Allegro PCB Librarian Part Developer Tool

You can create, fracture, and edit schematic symbols for your designs using the Cadence Allegro PCB Librarian Part Developer tool. Symbols designed in the Cadence Allegro PCB Librarian Part Developer tool can be split or fractured into several functional blocks called slots, allowing multiple smaller part fractures to exist on the same schematic page or across multiple pages.

15.5.1.1.1 Cadence Allegro PCB Librarian Part Developer Tool in the Design Flow



To run the Cadence Allegro PCB Librarian Part Developer tool, you must open a Cadence Allegro Design Entry HDL project in the Cadence Allegro Project Manager tool. To open the Cadence Allegro PCB Librarian Part Developer tool, on the Flows menu, click **Library Management**, and then click **Part Developer**.

Related Links

FPGA-to-PCB Design Flow on page 262

15.5.1.1.2 Import and Export Wizard

After starting the Cadence Allegro PCB Librarian Part Developer tool, use the **Import and Export**wizard to import your pin assignments from the Quartus Prime software.

Note: Intel recommends using your PCB Librarian Expert license file. To point to your PCB Librarian Expert license file, on the File menu, click **Change Product** and then select the correct product license.

To access the Import and Export wizard, follow these steps:

1. On the File menu, click **Import and Export**.
2. Select **Import ECO-FPGA**, and then click **Next**.
3. In the **Select Source** page of the **Import and Export** wizard, specify the following settings:
   a. In the **Vendor** list, select **Altera**.
   b. In the **PnR Tool** list, select **quartusII**.
   c. In the **PR File** box, browse to select the **.pin** in your Quartus Prime project directory.
   d. Click **Simulation Options** to select simulation input files.

e. Click **Next.**

4. In the **Select Destination** dialog box, specify the following settings:
   a. Under **Select Component**, click **Generate Custom Component** to create a new component in a library,

      or

      Click **Use standard component** to base your symbol on an existing component.

      Note: Intel recommends creating a new component if you previously created a generic component for an FPGA device. Generic components can cause some problems with your design. When you create a new component, you can place your pin and signal assignments from the Quartus Prime software on this component and reuse the component as a base when you have a new FPGA design.

   b. In the **Library** list, select an existing library. You can select from the cells in the selected library. Each cell represents all the symbol versions and part fractures for a particular part. In the **Cell** list, select the existing cell to use as a base for your part.

   c. In the **Destination Library** list, select a destination library for the component. Click **Next**.

   d. Review and edit the assignments you import into the Cadence Allegro PCB Librarian Part Developer tool based on the data in the **.pin** and then click **Finish**. The location of each pin is not included in the **Preview of Import Data** page of the **Import and Export** wizard, but input pins are on the left side of the created symbol, output pins on the right, power pins on the top, and ground pins on the bottom.

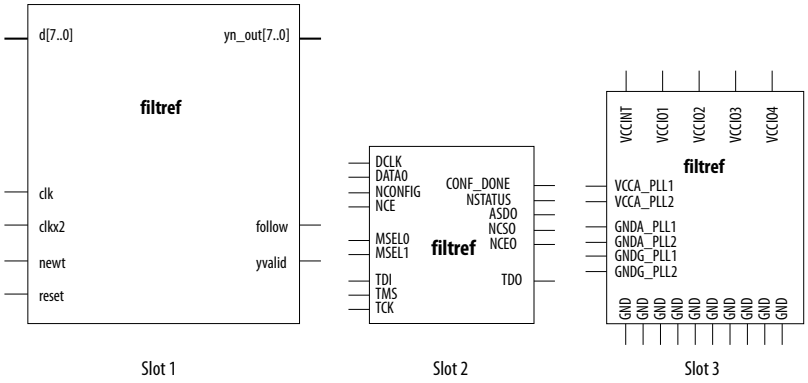
15.5.1.1.3 Editing and Fracturing Symbol

After creating your new symbol in the Cadence Allegro PCB Librarian Part Developer tool, you can edit the symbol graphics, fracture the symbol into multiple slots, and add or change package or symbol properties.

The Part Developer Symbol Editor contains many graphical tools to edit the graphics of a particular symbol. To edit the symbol graphics, select the symbol in the cell hierarchy. The **Symbol Pins** tab appears. You can edit the preview graphic of the symbol in the **Symbol Pins** tab.

Fracturing a Cadence Allegro PCB Librarian Part Developer package into separate symbol slots is useful for FPGA designs. A single symbol for most FPGA packages might be too large for a single schematic page. Splitting the part into separate slots allows you to organize parts of the symbol by function, creating cleaner circuit schematics. For example, you can create one slot for an I/O symbol, a second slot for a JTAG symbol, and a third slot for a power/ground symbol.

Figure 109. Splitting a Symbol into Multiple Slots



- This diagram represents a Cyclone device with JTAG or passive serial (PS) mode configuration option settings. Symbols created for other devices or other configuration modes may have dif erent sets of configuration pins, but can be fractured in a similar manner.
- The power/ground slot shows only a representation of power and ground pins because the device contains a large number of power and ground pins.

To fracture a part into separate slots, or to modify the slot locations of pins on parts fractured in the Cadence Allegro PCB Librarian Part Developer tool, follow these steps:

1. Start the Cadence Allegro Design Project Manager.
2. On the Flows menu, click **Library Management**.
3. Click **Part Developer**.
4. Click the name of the package you want to change in the cell hierarchy.
5. Click **Functions/Slots**. If you are not creating new slots but want to change the slot location of some pins, proceed to Step 6. If you are creating new slots, click **Add**. A dialog box appears, allowing you to add extra symbol slots. Set the number of extra slots you want to add to the existing symbol, not the total number of desired slots for the part. Click **OK**.
6. Click **Distribute Pins**. Specify the slot location for each pin. Use the checkboxes in each column to move pins from one slot to another. Click **OK**.
7. After distributing the pins, click the **Package Pin** tab and click **Generate Symbol(s)**.
8. Select whether to create a new symbol or modify an existing symbol in each slot. Click **OK**.

   The newly generated or modified slot symbols appear as separate symbols in the cell hierarchy. Each of these symbols can be edited individually.

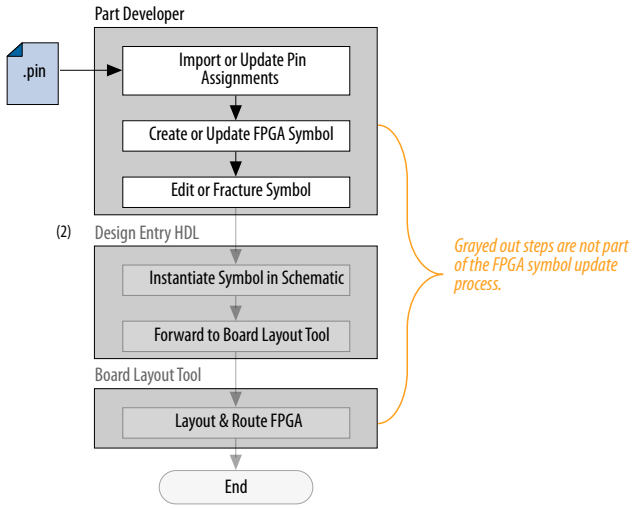
Caution: The Cadence Allegro PCB Librarian Part Developer tool allows you to remap pin assignments in the **Package Pin** tab of the main Cadence Allegro PCB Librarian Part Developer window. If signals remap to different pins in the Cadence Allegro PCB Librarian Part Developer tool, the changes reflect only in regenerated symbols for use in your schematics. You cannot transfer pin assignment changes to the Quartus Prime software from the Cadence Allegro PCB Librarian Part Developer tool, which creates a potential mismatch of the schematic symbols and assignments in the FPGA design. If pin assignment changes are necessary, make the changes in the Quartus Prime Pin Planner instead of the Cadence Allegro PCB Librarian Part Developer tool, and update the symbol as described in the following sections.

For more information about creating, editing, and organizing component symbols with the Cadence Allegro PCB Librarian Part Developer tool, refer to the Part Developer Help.

15.5.1.1.4 Updating FPGA Symbols

As the design process continues, you must make logic changes in the Quartus Prime software, placing signals on different pins after recompiling the design, or use the Quartus Prime Pin Planner to make changes manually. The board designer can request such changes to improve the board routing and layout. To ensure signals connect to the correct pins on the FPGA, you must carry forward these types of changes to the circuit schematic and board layout tools. Updating the **`.pin`** in the Quartus Prime software facilitates this flow.

Figure 110. Updating the FPGA Symbol in the Design Flow



To update the symbol using the Cadence Allegro PCB Librarian Part Developer tool after updating the **.pin**, follow these steps:

1. On the File menu, click **Import and Export**. The Import and Export wizard appears.
2. In the list of actions to perform, select **Import ECO - FPGA**. Click **Next**. The **Select Source** dialog box appears.
3. Select the updated source of the FPGA assignment information. In the **Vendor** list, select **Altera**. In the **PnR Tool** list, select **quartusII**. In the **PR File** field, click **browse** to specify the updated **.pin** in your Quartus Prime project directory. Click **Next**. The Select Destination window appears.
4. Select the source component and a destination cell for the updated symbol. To create a new component based on the updated pin assignment data, select **Generate Custom Component**. Selecting **Generate Custom Component** replaces the cell listed under the **Specify Library and Cell** name header with a new, nonfractured cell. You can preserve these edits by selecting **Use standard component and select the existing library and cell**. Select the destination library for the component and click **Next**. The **Preview of Import Data** dialog box appears.
5. Make any additional changes to your symbol. Click **Next**. A list of ECO messages appears summarizing the changes made to the cell. To accept the changes and update the cell, click **Finish**.
6. The main Cadence Allegro PCB Librarian Part Developer window appears. You can edit, fracture, and generate the updated symbols as usual from the main Cadence Allegro PCB Librarian Part Developer window.

Note: If the Cadence Allegro PCB Librarian Part Developer tool is not set up to point to your PCB Librarian Expert license file, an error message appears in red at the bottom of the message text window of the Part Developer when you select the **Import and Export** command. To point to your PCB Librarian Expert license, on the File menu, click **Change Product**, and select the correct product license.

Related Links

FPGA-to-PCB Design Flow on page 262

15.5.2 Instantiating the Symbol in the Cadence Allegro Design Entry HDL Software

To instantiate the symbol in your Cadence Allegro Design Entry HDL schematic after saving the new symbol in the Cadence Allegro PCB Librarian Part Developer tool, follow these steps:

1. In the Cadence Allegro Project Manager tool, switch to the board design flow.
2. On the Flows menu, click **Board Design**.
3. To start the Cadence Allegro Design Entry HDL software, click **Design Entry**.
4. To add the newly created symbol to your schematic, on the Component menu, click **Add**. The **Add Component** dialog box appears.
5. Select the new symbol library location, and select the name of the cell you created from the list of cells.

The symbol attaches to your cursor for placement in the schematic. To fracture the symbol into slots, right-click the symbol and choose **Version** to select one of the slots for placement in the schematic.

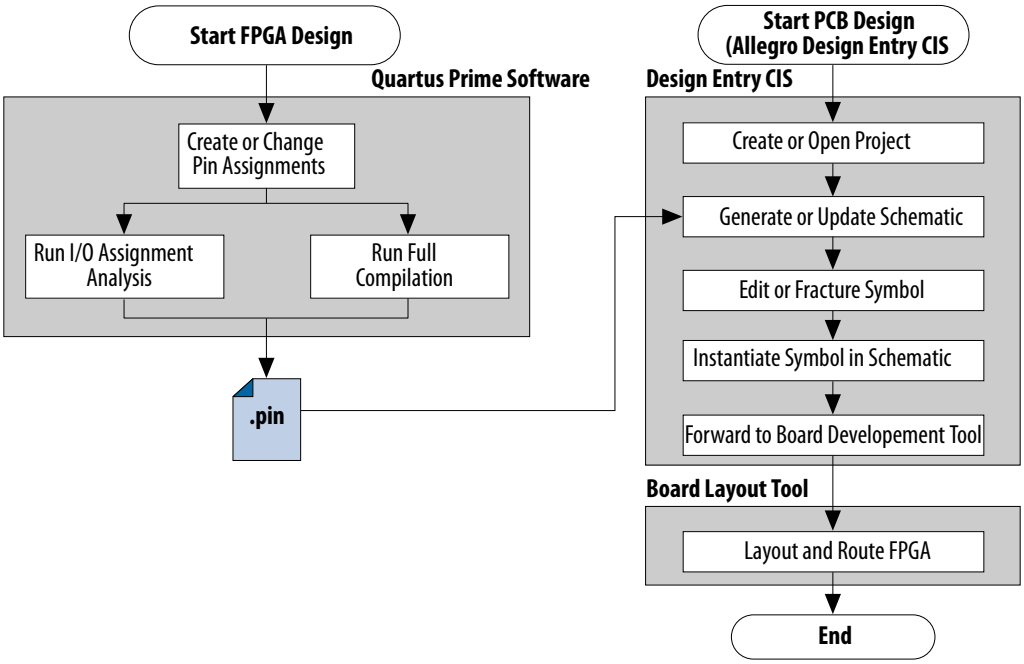
Related Links

www.cadence.com
Provides more information about the Cadence Allegro Design Entry HDL software, including licensing, support, usage, training, and product updates.

15.6 FPGA-to-Board Integration with Cadence Allegro Design Entry CIS Software

The Cadence Allegro Design Entry CIS software is a schematic capture tool (part of the Cadence 200 series design flow based on OrCAD Capture CIS). Use the Cadence Allegro Design Entry CIS software to create flat circuit schematics for all types of PCB design. You can also create hierarchical schematics to facilitate design reuse and team-based design using the Cadence Allegro Design Entry CIS software. With the Cadence Allegro Design Entry CIS software, the design flow from FPGA-to-board is unidirectional using only the **`.pin`** generated by the Quartus Prime software. You can only make signal and pin assignment changes in the Quartus Prime software. These changes reflect as updated symbols in a Cadence Allegro Design Entry CIS schematic project.

Figure 111. Design Flow with the Cadence Allegro Design Entry CIS Software



Note: Routing or pin assignment changes made in a board layout tool or a Cadence Allegro Design Entry CIS symbol cannot be back-annotated to the Quartus Prime software.

Related Links

- www.cadence.com

For more information about the Cadence Allegro Design Entry CIS software, including licensing, support, usage, training, and product updates.

- www.ema-eda.com
  For more information about the Cadence Allegro Design Entry CIS software, including licensing, support, usage, training, and product updates.

15.6.1 Creating a Cadence Allegro Design Entry CIS Project

The Cadence Allegro Design Entry CIS software has built-in support for creating schematic symbols using pin assignment information imported from the Quartus Prime software.

To create a new project in the Cadence Allegro Design Entry CIS software, follow these steps:

1. On the File menu, point to **New** and click **Project.** The New Project wizard starts.

   When you create a new project, you can select the PC Board wizard, the Programmable Logic wizard, or a blank schematic.
2. Select the PC Board wizard to create a project where you can select which part libraries to use, or select a blank schematic.

The Programmable Logic wizard only builds an FPGA logic design in the Cadence Allegro Design Entry CIS software.

Your new project is in the specified location and consists of the following files:

- OrCAD Capture Project File (**.opj**)
- Schematic Design File (**.dsn**)

15.6.2 Generating a Part

After you create a new project or open an existing project in the Cadence Allegro Design Entry CIS software, you can generate a new schematic symbol based on your Quartus Prime FPGA design. You can also update an existing symbol. The Cadence Allegro Design Entry CIS software stores component symbols in OrCAD Library File (**.olb**). When you place a symbol in a library attached to a project, it is immediately available for instantiation in the project schematic.

You can add symbols to an existing library or you can create a new library specifically for the symbols generated from your FPGA designs. To create a new library, follow these steps:

1. On the File menu, point to **New** and click **Library** in the Cadence Allegro Design Entry CIS software to create a default library named **library1.olb**. This library appears in the **Library** folder in the Project Manager window of the Cadence Allegro Design Entry CIS software.
2. To specify a desired name and location for the library, right-click the new library and select **Save As**. Saving the new library creates the library file.

15.6.3 Generating Schematic Symbol

You can now create a new symbol to represent your FPGA design in your schematic.

To generate a schematic symbol, follow these steps:

1. Start the Cadence Allegro Design Entry CIS software.
2. On the Tools menu, click **Generate Part**. The **Generate Part** dialog box appears.
3. To specify the **.pin** from your Quartus Prime design, in the **Netlist/source file type** field, click **Browse**.
4. In the **Netlist/source file type** list, select **Altera Pin File**
5. Type the new part name.
6. Specify the **Destination part library** for the symbol. Failing to select an existing library for the part creates a new library with a default name that matches the name of your Cadence Allegro Design Entry CIS project.
7. To create a new symbol for this design, select **Create new part**. If you updated your **.pin** in the Quartus Prime software and want to transfer any assignment changes to an existing symbol, select **Update pins on existing part in library**.
8. Select any other desired options and set **Implementation type** to **<none>**. The symbol is for a primitive library part based only on the **.pin** and does not require special implementation. Click **OK**.
9. Review the Undo warning and click **Yes** to complete the symbol generation.

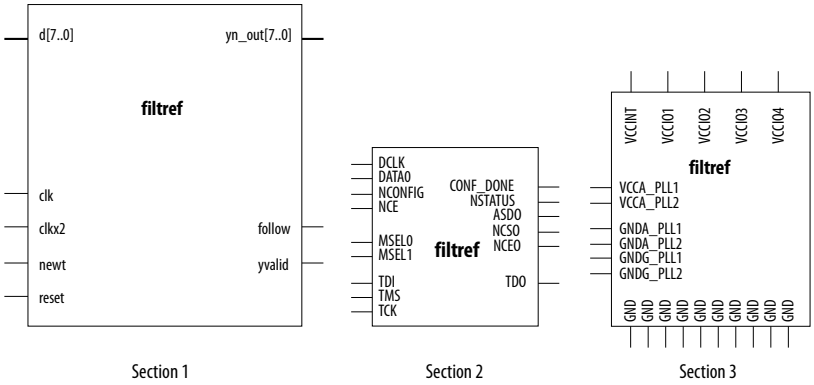
   You can locate the generated symbol in the selected library or in a new library found in the **Outputs** folder of the design in the Project Manager window. Double-click the name of the new symbol to see its graphical representation and edit it manually using the tools available in the Cadence Allegro Design Entry CIS software.

   Note: For more information about creating and editing symbols in the Cadence Allegro Design Entry CIS software, refer to the Help in the software.

15.6.4 Splitting a Part

After saving a new symbol in a project library, you can fracture the symbol into multiple parts called sections. Fracturing a part into separate sections is useful for FPGA designs. A single symbol for most FPGA packages might be too large for a single schematic page. Splitting the part into separate sections allows you to organize parts of the symbol by function, creating cleaner circuit schematics. For example, you can create one slot for an I/O symbol, a second slot for a JTAG symbol, and a third slot for a power/ground symbol.

Figure 112. Splitting a Symbol into Multiple Sections

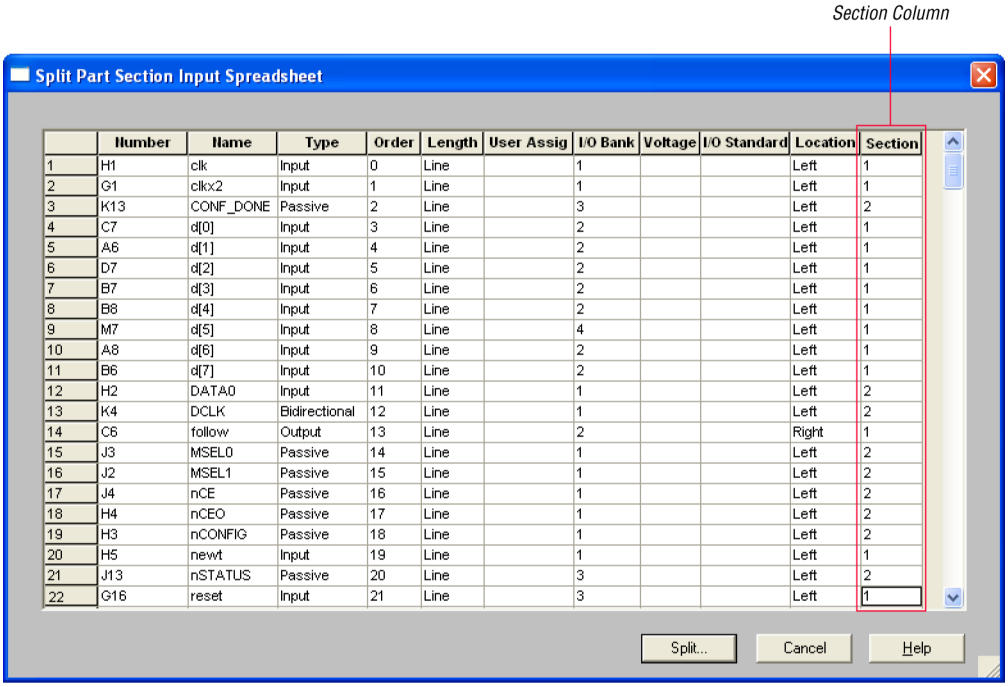


- This diagram represents a Cyclone device with JTAG or passive serial (PS) mode configuration option settings. Symbols created for other devices or other configuration modes might have dif erent sets of configuration pins, but can be fractured in a similar manner.
- The power/ground section shows only a representation of power and ground pins because the device contains a high number of power and ground pins.

Note: Although symbol generation in the Design Entry CIS software refers to symbol fractures as sections, other tools use different names to refer to symbol fractures.

To split a part into sections, select the part in its library in the Project Manager window of the Cadence Allegro Design Entry CIS software. On the Tools menu, click **Split Part** or right-click the part and choose **Split Part**. The **Split Part Section Input Spreadsheet** appears.

Figure 113. Split Part Section Input Spreadsheet



	Number	Name	Type	Order	Length	User Assig	I/O Bank	Voltage	I/O Standard	Location	Section
1	H1	clk	Input	0	Line		1			Left	1
2	G1	clkx2	Input	1	Line		1			Left	1
3	K13	CONF_DONE	Passive	2	Line		3			Left	2
4	C7	d[0]	Input	3	Line		2			Left	1
5	A6	d[1]	Input	4	Line		2			Left	1
6	D7	d[2]	Input	5	Line		2			Left	1
7	B7	d[3]	Input	6	Line		2			Left	1
8	B8	d[4]	Input	7	Line		2			Left	1
9	M7	d[5]	Input	8	Line		4			Left	1
10	A8	d[6]	Input	9	Line		2			Left	1
11	B6	d[7]	Input	10	Line		2			Left	1
12	H2	DATA0	Input	11	Line		1			Left	2
13	K4	DCLK	Bidirectional	12	Line		1			Left	2
14	C6	follow	Output	13	Line		2			Right	1
15	J3	MSEL0	Passive	14	Line		1			Left	2
16	J2	MSEL1	Passive	15	Line		1			Left	2
17	J4	nCE	Passive	16	Line		1			Left	2
18	H4	nCEO	Passive	17	Line		1			Left	2
19	H3	nCONFIG	Passive	18	Line		1			Left	2
20	H5	newt	Input	19	Line		1			Left	1
21	J13	nSTATUS	Passive	20	Line		3			Left	2
22	G16	reset	Input	21	Line		3			Left	1

Each row in the spreadsheet represents a pin in the symbol. The **Section** column indicates the section of the symbol to which each pin is assigned. You can locate all pins in a new symbol in section 1. You can change the values in the **Section** column to assign pins to various sections of the symbol. You can also specify the side of a section on the location of the pin by changing the values in the **Location** column. When you are ready, click **Split**. A new symbol appears in the same library as the original with the name <original part name>_**Split1.**

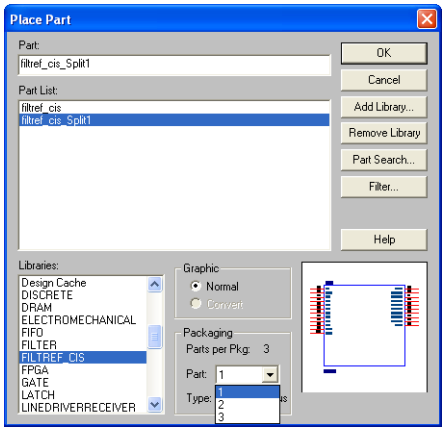
View and edit each section individually. To view the new sections of the part, double-click the part. The Part Symbol Editor window appears and the first section of the part displays for editing. On the View menu, click **Package** to view thumbnails of all the part sections. To edit the section of the symbol, double-click the thumbnail.

For more information about splitting parts into sections and editing symbol sections in the Cadence Allegro Design Entry CIS software, refer to the Help in the software.

15.6.5 Instantiating a Symbol in a Design Entry CIS Schematic

After saving a new symbol in a library in your Cadence Allegro Design Entry CIS project, you can instantiate the new symbol on a page in your schematic. Open a schematic page in the Project Manager window of the Cadence Allegro Design Entry CIS software. To add the newly created symbol to your schematic on the schematic page, on the Place menu, click **Part**. The **Place Part** dialog box appears.

Figure 114. Place Part Dialog Box



Select the new symbol library location and the newly created part name. If you select a part that is split into sections, you can select the section to place from the **Part** pop-up menu. Click **OK**. The symbol attaches to your cursor for placement in the schematic. To place the symbol, click on the schematic page.

For more information about using the Cadence Allegro Design Entry CIS software, refer to the Help in the software.

15.6.6 Intel Libraries for the Cadence Allegro Design Entry CIS Software

Intel provides downloadable **`.olb`** for many of its device packages. You can add these libraries to your Cadence Allegro Design Entry CIS project and update the symbols with the pin assignments contained in the **`.pin`** generated by the Quartus Prime software. You can use the downloaded library symbols as a base for creating custom

schematic symbols with your pin assignments that you can edit or fracture. This method increases productivity by reducing the amount of time it takes to create and edit a new symbol.

15.6.6.1 Using the Intel-provided Libraries with your Cadence Allegro Design Entry CIS Project

To use the Intel-provided libraries with your Cadence Allegro Design Entry CIS project, follow these steps:

1. Download the library of your target device from the Download Center page found through the Support page on the Altera website.
2. Create a copy of the appropriate **.olb** to maintain the original symbols. Place the copy in a convenient location, such as your Cadence Allegro Design Entry CIS project directory.
3. In the Project Manager window of the Cadence Allegro Design Entry CIS software, click once on the **Library** folder to select it. On the Edit menu, click **Project** or right-click the **Library** folder and choose **Add File** to select the copy of the downloaded **.olb** and add it to your project. You can locate the new library in the list of part libraries for your project.
4. On the Tools menu, click **Generate Part.** The **Generate Part** dialog box appears.
5. In the **Netlist/source file** field, click **Browse** to specify the **.pin** in your Quartus Prime design.
6. From the **Netlist/source file type** list, select **Altera Pin File**.
7. For **Part name**, type the name of the target device the same as it appears in the downloaded library file. For example, if you are using a device from the **CYCLONE06.OLB** library, type the part name to match one of the devices in this library such as `ep1c6f256`. You can rename the symbol in the Project Manager window after updating the part.
8. Set the **Destination part library** to the copy of the downloaded library you added to the project.
9. Select **Update pins on existing part in library**. Click **OK.**
10. Click **Yes**.

    The symbol is updated with your pin assignments. Double-click the symbol in the Project Manager window to view and edit the symbol. On the View menu, click **Package** if you want to view and edit other sections of the symbol. If the symbol in the downloaded library is fractured into sections, you can edit each section but you cannot further fracture the part. You can generate a new part without using the downloaded part library if you require additional sections.

    For more information about creating, editing, and fracturing symbols in the Cadence Allegro Design Entry CIS software, refer to the Help in the software.

15.7 Document Revision History

Table 63. Document Revision History

Date	Version	Changes
2016.10.31	16.1.0	• Implemented Intel rebranding.
2015.11.02	15.1.0	• Changed instances of *Quartus II* to *Quartus Prime*.
June 2014	14.0.0	Converted to DITA format.
June 2012	12.0.0	Removed survey link.
November 2011	10.0.2	Template update.
December 2010	10.0.1	Template update.
July 2010	10.0.0	• General style editing. • Removed Referenced Document Section. • Added a link to Help in "Performing Simultaneous Switching Noise (SSN) Analysis of Your FPGA" on page 9–5.
November 2009	9.1.0	• Added "Performing Simultaneous Switching Noise (SSN) Analysis of Your FPGA" on page 9–5. • General style editing. • Edited Figure 9–4 on page 9–10 and Figure 9–8 on page 9–16.
March 2009	9.0.0	• Chapter 9 was previously Chapter 7 in the 8.1 software release. • No change to content.
November 2008	8.1.0	Changed to 8-1/2 x 11 page size.
May 2008	8.0.0	Updated references.

Related Links

Altera Documentation Archive

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

16 Mentor Graphics PCB Design Tools Support

You can integrate the Mentor Graphics ® I/O Designer or DxDesigner PCB design tools into the Quartus Prime design flow. This combination provides a complete FPGA-to-board design workflow.

With today's large, high-pin-count and high-speed FPGA devices, good and correct PCB design practices are essential to ensure correct system operation. The PCB design takes place concurrently with the design and programming of the FPGA. The FPGA or ASIC designer initially creates signal and pin assignments, and the board designer must correctly transfer these assignments to the symbols in their system circuit schematics and board layout. As the board design progresses, Intel recommends reassigning pins to optimize the PCB layout. Ensure that you inform the FPGA designer of the pin reassignments so that the new assignments are included in an updated placement and routing of the design.

The Mentor Graphics I/O Designer software allows you to take advantage of the full FPGA symbol design, creation, editing, and back-annotation flow supported by the Mentor Graphics tools.

This chapter covers the following topics:

- Mentor Graphics and Intel software integration flow
- Generating supporting files
- Adding Quartus Prime I/O assignments to I/O Designer
- Updating assignment changes between the I/O Designer the Quartus Prime software
- Generating I/O Designer symbols
- Creating DxDesigner symbols from the Quartus Prime output files

This chapter is intended for board design and layout engineers who want to start the FPGA board integration while the FPGA is still in the design phase. Alternatively, the board designer can plan the FPGA pin-out and routing requirements in the Mentor Graphics tools and pass the information back to the Quartus Prime software for placement and routing. Part librarians can also benefit from this chapter by learning how to use output from the Quartus Prime software to create new library parts and symbols.

The procedures in this chapter require the following software:

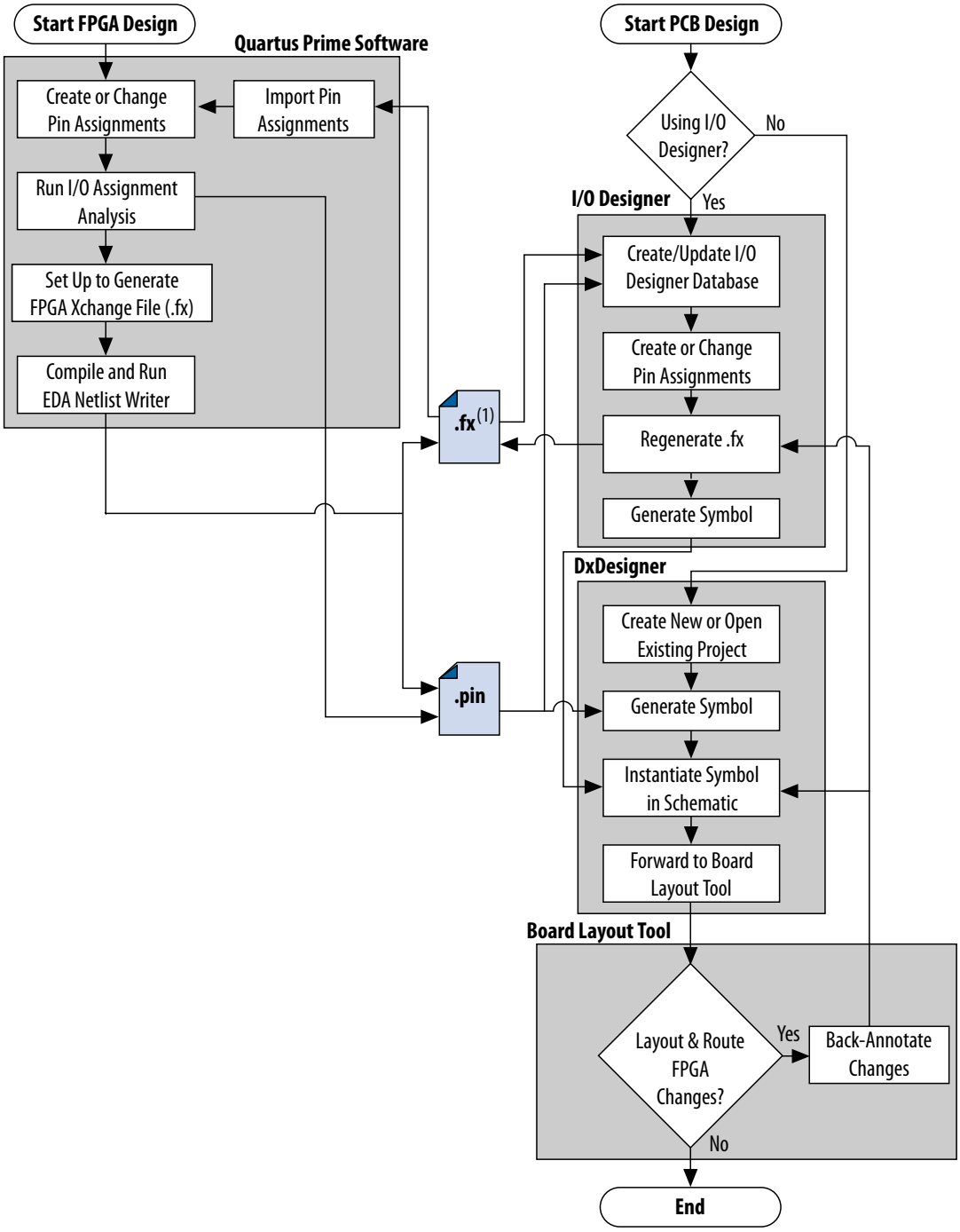
- The Quartus Prime software version 5.1 or later
- DxDesigner software version 2004 or later
- Mentor Graphics I/O Designer software (optional)

Note: To obtain and license the Mentor Graphics tools and for product information, support, and training, refer to the Mentor Graphics website.

16.1 FPGA-to-PCB Design Flow

You can create a design flow integrating an Intel FPGA design from the Quartus Prime software, and a circuit schematic in the DxDesigner software.

Figure 115. Design Flow with and Without the I/O Designer Software



Note: The Quartus Prime software generates the **.fx** in the output directory you specify in the **Board-Level** page of the **Settings** dialog box. However, the Quartus Prime software and the I/O Designer software can import pin assignments from an **.fx** located in any directory. Use a backup **.fx** to prevent overwriting existing assignments or importing invalid assignments.

To integrate the I/O Designer into your design flow, follow these steps:

1. In the Quartus Prime software, click **Assignments ➤ Settings ➤ EDA Tool Settings ➤ Board-Level** to specify settings for **.fx** symbol file generation.
2. Compile your design to generate the **.fx** and Pin-Out File (**.pin**) in the Quartus Prime project directory.
3. Create a board design with the DxDesigner software and the I/O Designer software by performing the following steps:
   a. Create a new I/O Designer database based on the **.fx** and the **.pin** files.
   b. In the I/O Designer software, make adjustments to signal and pin assignments.
   c. Regenerate the **.fx** in the I/O Designer software to export the I/O Designer software changes to the Quartus Prime software.
   d. Generate a single or fractured symbol for use in the DxDesigner software.
   e. Add the symbol to the **sym** directory of a DxDesigner project, or specify a new DxDesigner project with the new symbol.
   f. Instantiate the symbol in your DxDesigner schematic and export the design to the board layout tool.
   g. Back-annotate pin changes created in the board layout tool to the DxDesigner software and back to the I/O Designer software and the Quartus Prime software.
4. Create a board design with the DxDesigner software without the I/O Designer software by performing the following steps:
   a. Create a new DxBoardLink symbol with the **Symbol** wizard and reference the **.pin** from the Quartus Prime software in an existing DxDesigner project.
   b. Instantiate the symbol in your DxDesigner schematic and export the design to a board layout tool.

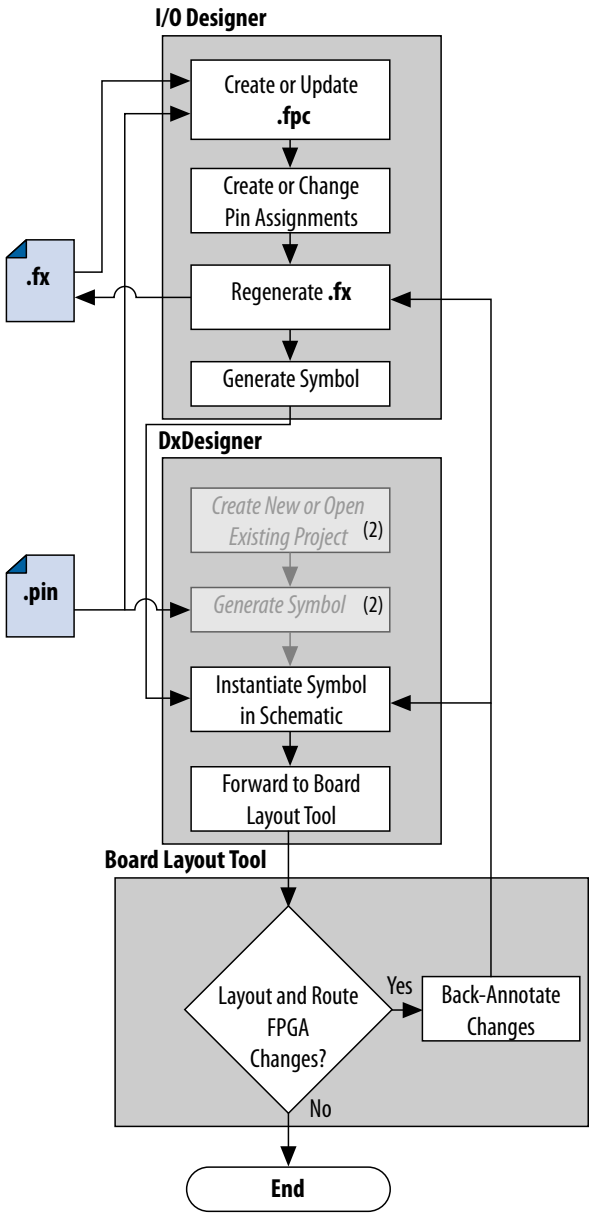
Note: You can update these symbols with design changes with or without the I/O Designer software. If you use the Mentor Graphics I/O Designer software and you change symbols with the DxDesigner software, you must reimport the symbols into I/O Designer to avoid overwriting your symbol changes.

16.2 Integrating with I/O Designer

You can integrate the Mentor Graphics I/O Designer software into the Quartus Prime design flow. Pin and signal assignment changes can be made anywhere in the design flow with either the Quartus Prime Pin Planner or the I/O Designer software. The I/O Designer software facilitates moving these changes, as well as synthesis, placement, and routing changes, between the Quartus Prime software, an external synthesis tool (if used), and a schematic capture tool such as the DxDesigner software.

This section describes how to use the I/O Designer software to transfer pin and signal assignment information to and from the Quartus Prime software with an **.fx**, and how to create symbols for the DxDesigner software.

Figure 116. I/O Designer Design Flow



Note: (2) DxDesigner software-specific steps in the design flow are not part of the I/O Designer flow.

16.2.1 Generating Pin Assignment Files

You transfer I/O pin assignments from the Quartus Prime software to the Mentor Graphics PCB tools by generating optional **.pin** and **.fx** files during Quartus Prime compilation. These files contain pin assignment information for use in other tools.

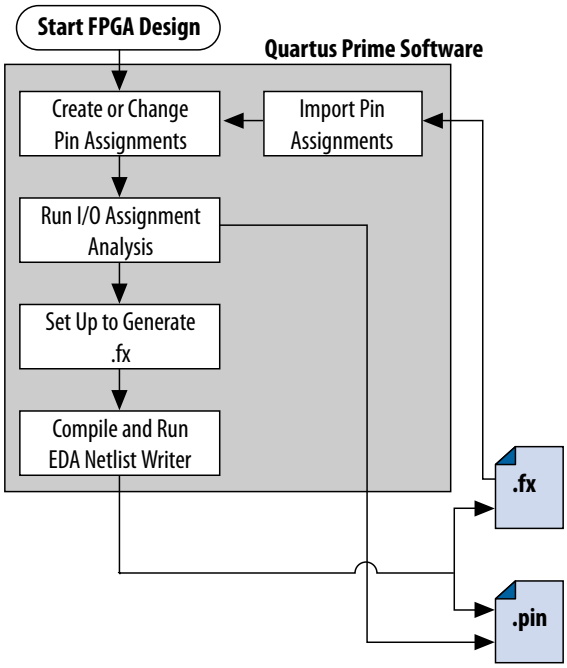
Click **Assignments ➤ Settings ➤ Board-Level** to specify settings for optional PCB tool file generation. Click **Processing ➤ Start Compilation** to compile the design to generate the file(s) in the project directory.

The Quartus Prime-generated **.pin** contains the I/O pin name, number, location, direction, and I/O standard for all used and unused pins in the design. Click **Assignments ➤ Pin Planner** to modify I/O pin assignments. You cannot import pin assignment changes from a Mentor Graphics **.pin** into the Quartus Prime software.

The **.fx** is an input or output of either the Quartus Prime or I/O Designer software. You can generate an **.fx** in the Quartus Prime software for symbol generation in the Mentor Graphics I/O Designer software. A Quartus Prime **.fx** contains the pin name, number, location, direction, I/O standard, drive strength, termination, slew rate, IOB delay, and differential pins. An I/O Designer **.fx** additionally includes information about unused pins and pin set groups.

The I/O Designer software can also read from or update a Quartus Prime Settings File (**.qsf**). You can use the **.qsf** in the same way as use of the **.fx**, but pin swap group information does not transfer between I/O Designer and the Quartus Prime software. Use the **.fx** rather than the **.qsf** for transferring I/O assignment information.

Figure 117. Generating .pin and .fx files



16.2.2 I/O Designer Settings

You can directly export I/O Designer symbols to the DxDesigner software. To set options for integrating I/O Designer with Dx Designer, follow these steps:

1. Start the I/O Designer software.
2. Click **Tools ➤ Prefences**.
3. Click **Paths**, and then double-click the **DxDesigner executable file** path field to select the location of the DxDesigner application.
4. Click **Apply**.
5. Click **Symbol Editor**, and then click **Export**. In the Export type menu, under **General**, select **DxDesigner/PADS-Designer**.
6. Click **Apply**, and then click **OK**.
7. Click **File ➤ Properties**.
8. Click the **PCB Flow** tab, and then click **Path to a DxDesigner project directory**.
9. Click **OK**.
   If you do not have a new DxDesigner project in the Database wizard and a DxDesigner project, you must create a new database with the DxDesigner software, and then specify the project location in I/O Designer.

16.2.3 Transferring I/O Assignments

You can transfer Quartus Prime signal and pin assignments contained in **.pin** and **.fx** files into an I/O Designer database. Use the I/O Designer Database Wizard to create a new database incorporating the **.fx** and **.pin** files. You can also create a new, empty database and manually add the assignment information. If there is no available signal or pin assignment information, you can create an empty database containing only a selection of the target device. This technique is useful if you know the signals in your design and the pins you want to assign. You can subsequently transfer this information to the Quartus Prime software for placement and routing.

You may create a very simple I/O Designer database that includes only the **.pin** or **.fx** file information. However, when using only a **.pin**, you cannot import I/O assignment changes from I/O Designer back into the Quartus Prime software without also generating an **.fx**. If your I/O Designer database includes only **.fx** file information, the database may not contain all the available I/O assignment information. The Quartus Prime **.fx** file only lists assigned pins. The **.pin** lists all assigned and unassigned device pins. Use both the **.pin** and the **.fx** to produce the most complete set of I/O Designer database information.

To create a new I/O Designer database using the Database wizard, follow these steps;

1. Start the I/O Designer software. The **Welcome to I/O Designer** dialog box appears. Select **Wizard to create new database** and click **OK.**
   If the **Welcome to I/O Designer** dialog box does not appear, you can access the wizard through the menu. To access the wizard, click **File ➤ Database Wizard**.
2. Click **Next**. The **Define HDL source file** page appears

If no HDL files are available, or if the **.fx** contains your signal and pin assignments, you can skip Step 3 and proceed to Step 4.

3. If your design includes a Verilog HDL or VHDL file, you can add a top-level Verilog HDL or VHDL file in the I/O Designer software. Adding a file allows you to create functional blocks or get signal names from your design. You must create all physical pin assignments in I/O Designer if you are not using an **.fx** or a **.pin**. Click **Next**. The **Database Name** page appears.
4. In the **Database Name** page, type your database file name. Click **Next**. The Database Location window appears.
5. Add a path to the new or an existing database in the **Location** field, or browse to a database location. Click **Next**. The **FPGA flow** page appears.
6. In the Vendor menu, click **Altera**.
7. In the Tool/Library menu, click **Quartus Prime <version>** to select your version of the Quartus Prime software.

   Note: The Quartus Prime software version listed may not match your actual software version. If your version is not listed, select the latest version. If your target device is not available, the device may not yet be supported by the I/O Designer software.
8. Select the appropriate device family, device, package, and speed (if applicable), from the corresponding menus. Click **Next**. The **Place and route** page appears.
9. In the **FPGAX file name** field, type or browse to the backup copy of the **.fx** generated by the Quartus Prime software.
10. In the **Pin report file name** field, type or browse to the **.pin** generated by the Quartus Prime software. Click **Next**.
    You can also select a **.qsf** for update. The I/O Designer software can update the pin assignment information in the **.qsf** without affecting any other information in the file.

    Note: You can import a **.pin** without importing an **.fx**. The I/O Designer software does not generate a **.pin**. To transfer assignment information to the Quartus Prime software, select an additional file and file type. Intel recommends selecting an **.fx** in addition to a **.pin** for transferring all the assignment information in the **.fx** and **.pin** files. In some versions of the I/O Designer software, the standard file picker may incorrectly look for a **.pin** instead of an **.fx**. In this case, select **All Files (*.*)** from the **Save as type** list and select the file from the list.
11. On the **Synthesis** page, specify an external synthesis tool and a synthesis constraints file for use with the tool. If you do not use an external synthesis tool, click **Next**.
12. On the **PCB Flow** page, you can select an existing schematic project or create a new project as a symbol information destination.

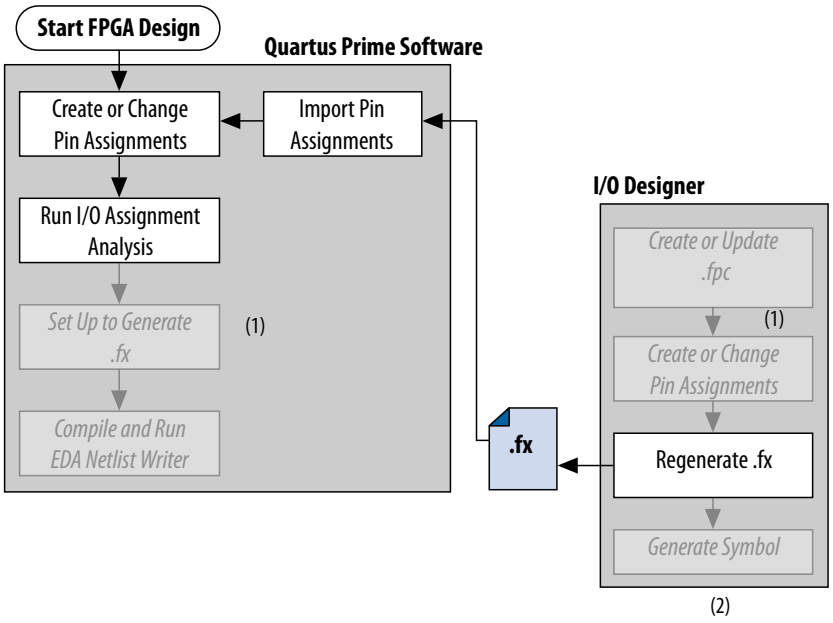
- To select an existing project, select Choose existing project and click Browse after the Project Path field. The Select project dialog box appears. Select the project.
- To create a new project, in the Select project dialog box, select Create new empty project. Type the project file name in the Name field and browse to the location where you want to save the file. Click OK.

13. If you have not specified a design tool to which you can send symbol information in the I/O Designer software, click **Advanced** in the **PCB Flow** page and select your design tool. If you select the DxDesigner software, you have the option to specify a Hierarchical Occurrence Attributes (**.oat**) file to import into the I/O Designer software. Click **Next** and then click **Finish** to create the database.Updating

16.2.4 Updating Quartus Prime with I/O Designer Pin Assignments

As you fine tune your board design in I/O Designer, changes to signal routing and layout are likely. You must import any routing and layout changes into the Quartus Prime software for accurate place and route to match the new pin-out. The I/O Designer tool supports this flow.

Figure 118. Importing I/O Designer Pin Assignments



To import I/O Designer pin assignments, follow these steps:

1. Make pin assignment changes directly in the I/O Designer software, or the software can automatically update changes made in a board layout tool that are back-annotated to a schematic entry program such as the DxDesigner software.
2. To update the **.fx** with the changes, click **Generate ➤ FPGA Xchange File**.
3. Open your Quartus Prime project.
4. Click **Assignments ➤ Import Assignments**.

5. (Optional) To preserve original assignments before import, turn on **Copy existing assignments into <project name>.qsf.bak** before importing before importing the **.fx**.
6. Select the **.fx** and click **Open**.
7. Click **OK**.

16.2.5 Generating Schematic Symbols in I/O Designer

Circuit board schematic creation is one of the first tasks required in the design of a new PCB. You can use the I/O Designer software to generate schematic symbols for your Quartus Prime FPGA design for use in the DXDesigner schematic entry tools. The I/O Designer software can generate symbols for use in various Mentor Graphics schematic entry tools, and can import changes back-annotated by board layout tools to update the database and update the Quartus Prime software with the **.fx**

Most FPGA devices contain hundreds of pins, requiring large schematic symbols that may not fit on a single schematic page. Symbol designs in the I/O Designer software can be split or fractured into various functional blocks, allowing multiple part fractures on the same schematic page or across multiple pages. In the DxDesigner software, these part fractures join together with the use of the HETER0 attribute.

You can use the I/O Designer **Symbol** wizard to quickly create symbols that you can subsequently refine. Alternatively, you can import symbols from another DXDesigner project, and then assign an FPGA to the symbol. To import symbols in the I/O Designer software, **File ➤ Import Symbol**.

I/O Designer symbols are either functional, physical (PCB), or both. Signals imported into the database, usually from Verilog HDL or VHDL files, are the basis of a functional symbol. No physical device pins must be associated with the signals to generate a functional symbol. This section focuses on board-level PCB symbols with signals directly mapped to physical device pins through assignments in either the Quartus Prime Pin Planner or in the I/O Designer database.

16.2.5.1 Generating Schematic Symbols

To create a symbol based on a selected Intel FPGA device, follow these steps:

1. Start the I/O Designer software.
2. Click **Symbol ➤ Symbol Wizard**.
3. In the **Symbol name** field, type the symbol name. The **DEVICE** and **PKG_TYPE** fields display the device and package information.

   Note: If **DEVICE** and **PKG_TYPE** are blank or incorrect, close the Symbol wizard and specify the correct device information (**File ➤ Properties ➤ FPGA Flow)**.
4. Under **Symbol type**, click **PCB**. Under **Use signals**, click **All**, then click **Next**.
5. Select fracturing options for your symbol. If you are using the Symbol wizard to edit a previously created fractured symbol, you must turn on **Reuse existing fractures** to preserve your current fractures. Select other options on this page as appropriate for your symbol. Click **Next**.
6. Select additional fracturing options for your symbol. Click **Next**.

7. Select the options for the appearance of the symbols. Click **Next**.
8. Define the information you want to label for the entire symbol and for individual pins. Click **Next**.
9. Add any additional signals and pins to the symbol. Click **Finish**.
   You can view your symbol and any fractures you created with the Symbol Editor. You can edit parts of the symbol, delete fractures, or rerun the Symbol wizard. When you modify pin assignments in I/O Designer database, I/O Designer symbols automatically reflect these changes. Modify assignments in the I/O Designer software by supplying and updated **.fx** from the Quartus Prime software, or by back-annotating changes in your board layout tool.

16.2.6 Exporting Schematic Symbols to DxDesigner

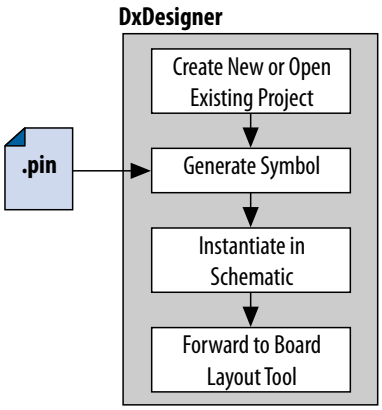
You can export your I/O Designer schematic symbols for to DxDesigner for further design entry work. To generate all fractures of a symbol, click **Generate ➤ All Symbols**. To generate only the currently displayed symbol, click **Generate ➤ Current Symbol Only**. The DxDesigner project **/sym** directory preserves each symbol in the database as a separate file. You can instantiate the symbols in your DxDesigner schematics.

16.3 Integrating with DxDesigner

You can integrate the Mentor Graphics DxDesigner schematic capture tool into the Quartus Prime design flow. Use DxDesigner to create flat circuit schematics or to create hierarchical schematics that facilitate design reuse and a team-based design for all PCB types. Use DxDesigner in conjunction with I/O Designer software for a complete FPGA I/O and PCB design flow.

If you use DxDesigner without the I/O Designer software, the design flow is one-way, using only the **.pin** generated by the Quartus Prime software. You can only make signal and pin assignment changes in the Quartus Prime software. You cannot back-annotate changes made in a board layout tool or in a DxDesigner symbol to the Quartus Prime software.

Figure 119. DxDesigner-only Flow (without I/O Designer)



16.3.1 DxDesigner Project Settings

DxDesigner new projects automatically create FPGA symbols by default. However, if you are using the I/O Designer with DxDesigner, you must enable DxBoardLink Flow options for integration with the I/O Designer software. To enable the DxBoardLink flow design configuration when creating a new DxDesigner project, follow these steps:

1. Start the DxDesigner software.
2. Click **File ➤ New**, and then click the **Project** tab.
3. Click **More**. Turn on **DxBoardLink**. To enable the DxBoardLink Flow design configuration for an existing project, click **Design Configurations** in the Design Configuration toolbar and turn on **DxBoardLink**.

16.3.2 Creating Schematic Symbols in DxDesigner

You can create schematic symbols in the DxDesigner software manually or with the Symbol wizard. The DxDesigner Symbol wizard is similar to the I/O Designer Symbol wizard, but with fewer fracturing options. The DxDesigner Symbol wizard creates, fractures, and edits FPGA symbols based on the specified Intel device. To create a symbol with the Symbol wizard, follow these steps;

1. Start the DxDesigner software.
2. Click **Symbol Wizard** in the toolbar.
3. Type the new symbol name in the name field and click **OK**.
4. Specify creation of a new symbol or modification of an existing symbol. To modify an existing symbol, specify the library path or alias, and select the existing symbol. To create a new symbol, select DxBoardLink for the symbol source. The DxDesigner block type defaults to Module because the FPGA design does not have an underlying DxDesigner schematic. Choose whether or not to fracture the symbol. Click **Next**.
5. Type a name for the symbol, an overall part name for all the symbol fractures, and a library name for the new library created for this symbol. By default, the part and library names are the same as the symbol name. Click **Next**.
6. Specify the appearance of the generated symbol and how itthe grid you have set in your DxDesigner project schematic. After making your selections. Click **Next**.
7. In the **FPGA vendor** list, select **Intel Quartus**. In the **Pin-Out file to import** field, select the **.pin** from your Quartus Prime project directory. You can also specify Fracturing Scheme, Bus pin, and Power pin options. Click **Next**.
8. Select to create or modify symbol attributes for use in the DxDesigner software. Click **Next**.
9. On the **Pin Settings** page, make any final adjustments to pin and label location and information. Each tabbed spreadsheet represents a fracture of your symbol. Click **Save Symbol**.
   After creating the symbol, you can examine and place any fracture of the symbol in your schematic. You can locate separate files of all the fractures you created in the library you specified or created in the **/sym** directory in your DxDesigner project. You can add the symbols to your schematics or you can manually edit the symbols or with the Symbol wizard.

16.4 Scripting API

The I/O Designer software includes a command line Tcl interpreter. All commands input through the I/O Designer GUI translate into Tcl commands run by the tool. You can run individual Tcl commands or scripts in the I/O Designer Console window, rather than using the GUI.

You can use the following Tcl commands to control I/O Designer.

- `set_fpga_xchange_file` <*file name*>—specifies the **.fx** from which the I/O Designer software updates assignments.
- `update_from_fpga_xchange_file`—updates the I/O Designer database with assignment updates from the currently specified **.fx**.
- `generate_fpga_xchange_file`—updates the **.fx** with I/O Designer software changes for transfer back into the Quartus Prime software.
- `set_pin_report_file -quartus_pin` <*file name*>—imports assignment data from a Quartus Prime software .pin file.
- `symbolwizard`—runs the I/O Designer Symbol wizard.
- `set_dx_designer_project -path` <*path*>

16.5 Document Revision History

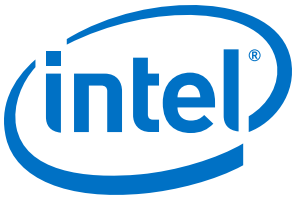
Table 64. Document Revision History

Date	Version	Changes
2016.10.31	16.1.0	• Implemented Intel rebranding.
2015.11.02	15.1.0	• Changed instances of *Quartus II* to *Quartus Prime*.
2014.06.30	14.0.0	• Replaced MegaWizard Plug-In Manager information with IP Catalog. • Added standard information about upgrading IP cores. • Added standard installation and licensing information. • Removed outdated device support level information. IP core device support is now available in IP Catalog and parameter editor.
June 2012	12.0.0	• Removed survey link.
December 2010	10.1.0	• Changed to new document template.

Related Links

Altera Documentation Archive

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



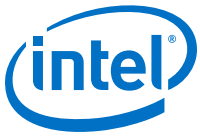
Intel® Quartus® Prime Pro Edition Handbook Volume 3: Verification

QPP5V3
2017.05.08


Subscribe


Send Feedback

Contents



1 Simulating Intel FPGA Designs

This document describes simulating designs that target Intel FPGA devices. Simulation verifies design behavior before device programming. The Quartus® Prime software supports RTL- and gate-level design simulation in supported EDA simulators. Simulation involves setting up your simulator working environment, compiling simulation model libraries, and running your simulation.

1.1 Simulator Support

The Quartus Prime software supports specific EDA simulator versions for RTL and gate-level simulation.

Table 1. Supported Simulators

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

1.2 Simulation Levels

The Quartus Prime software supports RTL and gate-level simulation of IP cores in supported EDA simulators.

Table 2. Supported Simulation Levels

Simulation Level	Description	Simulation Input
RTL	Cycle-accurate simulation using Verilog HDL, SystemVerilog, and VHDL design source code with simulation models provided by Intel and other IP providers.	• Design source/testbench • Intel simulation libraries • Intel FPGA IP plain text or IEEE encrypted RTL models • IP simulation models • Intel FPGA IP functional simulation models

continued...

Simulation Level	Description	Simulation Input
		• Intel FPGA IP bus functional models • Qsys Pro-generated models • Verification IP
Gate-level functional	Simulation using a post-synthesis or post-fit functional netlist testing the post-synthesis functional netlist, or post-fit functional netlist.	• Testbench • Intel simulation libraries • Post-synthesis or post-fit functional netlist • Intel FPGA IP bus functional models

1.3 HDL Support

The Quartus Prime software provides the following HDL support for EDA simulators.

Table 3. HDL Support

Language	Description
VHDL	• For VHDL RTL simulation, compile design files directly in your simulator. You must also compile simulation models from the Intel FPGA simulation libraries and simulation models for the IP cores in your design. Use the Simulation Library Compiler to compile simulation models. • For gate-level simulation, the EDA Netlist Writer generates a synthesized design netlist VHDL Output File (`.vho`). Compile the `.vho` in your simulator. You may also need to compile models from the Intel FPGA simulation libraries. • IEEE 1364-2005 encrypted Verilog HDL simulation models are encrypted separately for each simulation vendor that the Quartus Prime software supports. To simulate the model in a VHDL design, you must have a simulator that is capable of VHDL/Verilog HDL co-simulation.
Verilog HDL -SystemVerilog	• For RTL simulation in Verilog HDL or SystemVerilog, compile your design files in your simulator. You must also compile simulation models from the Intel FPGA simulation libraries and simulation models for the IP cores in your design. Use the Simulation Library Compiler to compile simulation models. • For gate-level simulation, the EDA Netlist Writer generates a synthesized design netlist Verilog Output File (`.vo`). Compile the `.vo` in your simulator.
Mixed HDL	• If your design is a mix of VHDL, Verilog HDL, and SystemVerilog files, you must use a mixed language simulator. Choose the most convenient supported language for generation of Intel FPGA IP cores in your design. • Intel FPGA provides the entry-level ModelSim - Intel FPGA Edition software, along with precompiled Intel FPGA simulation libraries, to simplify simulation of Intel FPGA designs. Starting in version 15.0, the ModelSim - Intel FPGA Edition software supports native, mixed-language (VHDL/Verilog HDL/SystemVerilog) co-simulation of plain text HDL. If you have a VHDL-only simulator and need to simulate Verilog HDL modules and IP cores, you can either acquire a mixed-language simulator license from the simulator vendor, or use the ModelSim - Intel FPGA Edition software.
Schematic	You must convert schematics to HDL format before simulation. You can use the converted VHDL or Verilog HDL files for RTL simulation.

1.4 Simulation Flows

The Quartus Prime software supports various simulation flows.

Table 4. Simulation Flows

Simulation Flow	Description
Scripted Simulation Flows	Scripted simulation supports custom control of all aspects of simulation, such as custom compilation commands, or multi-pass simulation flows. Use a version-independent top-level simulation script that "sources" Quartus Prime-generated IP simulation setup scripts. The Quartus Prime software generates a combined simulator setup script for all IP cores, for each supported simulator.
Specialized Simulation Flows	Supports specialized simulation flows for specific design variations, including the following: • For simulation of example designs, refer to the documentation for the example design or to the IP core user guide. • For simulation of Qsys Pro designs, refer to *Creating a System with Qsys* or *Creating a System with Qsys Pro*. • For simulation of designs that include the Nios® II embedded processor, refer to *Simulating a Nios II Embedded Processor*.

Related Links

- IP User Guide Documentation
- Creating a System with Qsys
- Creating a System with Qsys Pro
- Simulating a Nios II Embedded Processor

1.5 Preparing for Simulation

Preparing for RTL or gate-level simulation involves compiling the RTL or gate-level representation of your design and testbench. You must also compile IP simulation models, models from the Intel FPGA simulation libraries, and any other model libraries required for your design.

1.5.1 Compiling Simulation Models

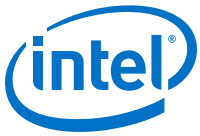
The Quartus Prime software includes simulation models for all Intel FPGA IP cores. These models include IP functional simulation models, and device family-specific models in the <Quartus Prime *installation path*>/eda/sim_lib directory. These models include IEEE encrypted Verilog HDL models for both Verilog HDL and VHDL simulation.

Before running simulation, you must compile the appropriate simulation models from the Quartus Prime simulation libraries using any of the following methods:

- Use the NativeLink feature to automatically compile your design, Intel FPGA IP, simulation model libraries, and testbench.
- Run the Simulation Library Compiler to compile all RTL and gate-level simulation model libraries for your device, simulator, and design language.
- Compile Quartus Prime simulation models manually with your simulator.

After you compile the simulation model libraries, you can reuse these libraries in subsequent simulations.

Note: The specified timescale precision must be within 1ps when using Quartus Prime simulation models.



Related Links

Quartus Prime Simulation Models
In Quartus Prime Pro Edition Help

1.6 Simulating Intel FPGA IP Cores

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

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

1.6.1 Generating IP Simulation Files

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

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

Table 5. Intel FPGA IP Simulation Files

File Type	Description	File Name
Simulator setup scripts	Vendor-specific scripts to compile, elaborate, and simulate Intel FPGA IP models and simulation model library files. Optionally, generate a simulator setup script for each vendor that combines the individual IP core scripts into one file. Source the combined script from your top-level simulation script to eliminate script maintenance.	<*my_dir*>/aldec/rivierapro_setup.tcl <*my_dir*>/cadence/ncsim_setup.sh <*my_dir*>/mentor/msim_setup.tcl <*my_dir*>/synopsys/vcs/vcs_setup.sh

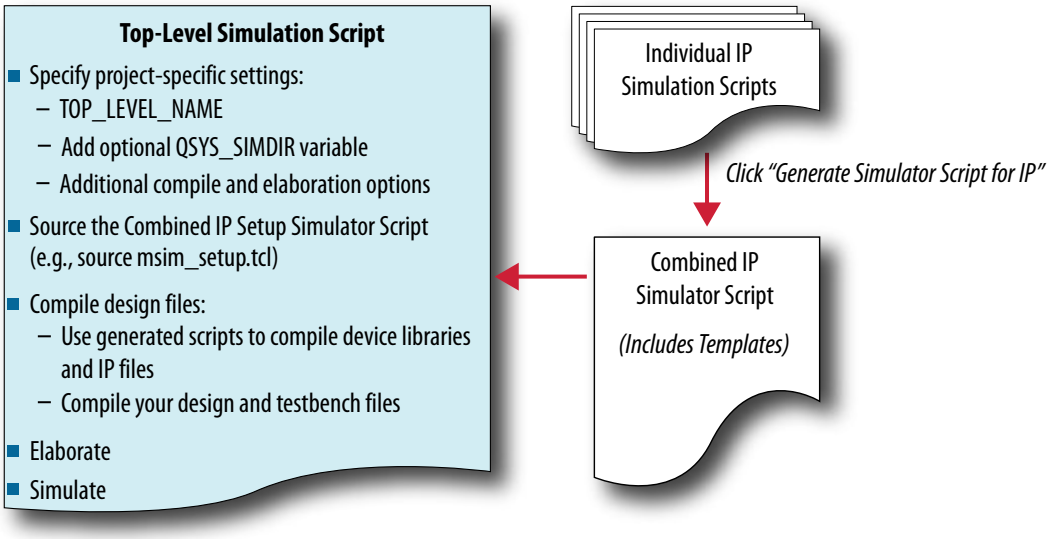
Note: Intel FPGA IP cores support a variety of cycle-accurate simulation models, including simulation-specific IP functional simulation models and encrypted RTL models, and plain text RTL models. The models support fast functional simulation of your IP core instance using industry-standard VHDL or Verilog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

1.6.2 Scripting IP Simulation

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

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

Figure 1. Incorporating Generated Simulator Setup Scripts into a Top-Level Simulation Script



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

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

1.6.2.1 Generating a Combined Simulator Setup Script

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

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

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

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

1. Generate, regenerate, or upgrade one or more IP core. Refer to *Generating IP Cores* or *Upgrading IP Cores*.
2. Click **Tools ➤ Generate Simulator Setup Script for IP** (or run the `ip-setup-simulation` utility). Specify the **Output Directory** and library compilation options. Click **OK** to generate the file. By default, the files generate into the */<project directory>/<simulator>/* directory using relative paths.
3. To incorporate the generated simulator setup script into your top-level simulation script, refer to the template section in the generated simulator setup script as a guide to creating a top-level script:
   a. Copy the specified template sections from the simulator-specific generated scripts and paste them into a new top-level file.
   b. Remove the comments at the beginning of each line from the copied template sections.
   c. Specify the customizations you require to match your design simulation requirements, for example:
      - Specify the `TOP_LEVEL_NAME` variable to the design's simulation top-level file. The top-level entity of your simulation is often a testbench that instantiates your design. Then, your design instantiates IP cores and/or Qsys or Qsys Pro systems. Set the value of `TOP_LEVEL_NAME` to the top-level entity.
      - If necessary, set the `QSYS_SIMDIR` variable to point to the location of the generated IP simulation files.
      - Compile the top-level HDL file (e.g. a test program) and all other files in the design.
      - Specify any other changes, such as using the `grep` command-line utility to search a transcript file for error signatures, or e-mail a report.
4. Re-run **Tools ➤ Generate Simulator Setup Script for IP** (or `ip-setup-simulation`) after regeneration of an IP variation.

Table 6. Simulation Script Utilities

Utility	Syntax

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

`ip-make-simscript` generates a combined simulation script for all IP cores that you specify on the command line. Specify one or more `.spd` files and an output directory in the command. Running the script compiles IP simulation models into various simulation libraries.	`ip-make-simscript`
	` --spd=<ipA.spd,ipB.spd>`
	` --output-directory=<directory>`

The following sections provide step-by-step instructions for sourcing each simulator setup script in your top-level simulation script.

1.6.2.2 Incorporating Simulator Setup Scripts from the Generated Template

You can incorporate generated IP core simulation scripts into a top-level simulation script that controls simulation of your entire design. After running `ip-setup-simulation` use the following information to copy the template sections and modify them for use in a new top-level script file.

1.6.2.2.1 Sourcing Aldec* Simulator Setup Scripts

Follow these steps to incorporate the generated Aldec simulation scripts into a top-level project simulation script.

1. The generated simulation script contains the following template lines. Cut and paste these lines into a new file. For example, `sim_top.tcl`.

```
# # Start of template
# # If the copied and modified template file is "aldec.do", run it as:
# # vsim -c -do aldec.do
# #
# # Source the generated sim script
# source rivierapro_setup.tcl
# # Compile eda/sim_lib contents first
# dev_com
# # Override the top-level name (so that elab is useful)
# set TOP_LEVEL_NAME top
# # Compile the standalone IP.
# com
# # Compile the user top-level
# vlog -sv2k5 ../../top.sv
# # Elaborate the design.
# elab
# # Run the simulation
# run
```

```
# # Report success to the shell
# exit -code 0
# # End of template
```

2. Delete the first two characters of each line (comment and space):

```
# Start of template
# If the copied and modified template file is "aldec.do", run it as:
# vsim -c -do aldec.do
#
# Source the generated sim script source rivierapro_setup.tcl
# Compile eda/sim_lib contents first dev_com
# Override the top-level name (so that elab is useful)
set TOP_LEVEL_NAME top
# Compile the standalone IP.
com
# Compile the user top-level vlog -sv2k5 ../../top.sv
# Elaborate the design.
elab
# Run the simulation
run
# Report success to the shell
exit -code 0
# End of template
```

3. Modify the TOP_LEVEL_NAME and compilation step appropriately, depending on the simulation's top-level file. For example:

```
set TOP_LEVEL_NAME sim_top
 vlog –sv2k5 ../../sim_top.sv
```

4. If necessary, add the QSYS_SIMDIR variable to point to the location of the generated IP simulation files. Specify any other changes that you require to match your design simulation requirements. The scripts offer variables to set compilation or simulation options. Refer to the generated script for details.
5. Run the new top-level script from the generated simulation directory:

```
vsim –c –do <path to sim_top>.tcl
```

1.6.2.2.2 Sourcing Cadence* Simulator Setup Scripts

Follow these steps to incorporate the generated Cadence IP simulation scripts into a top-level project simulation script.

1. The generated simulation script contains the following template lines. Cut and paste these lines into a new file. For example, `ncsim.sh`.

```
# # Start of template
# # If the copied and modified template file is "ncsim.sh", run it as:
# # ./ncsim.sh
# #
# # Do the file copy, dev_com and com steps
# source ncsim_setup.sh \
# SKIP_ELAB=1 \
# SKIP_SIM=1
#
# # Compile the top level module
# ncvlog -sv "$QSYS_SIMDIR/../top.sv"
#
# # Do the elaboration and sim steps
# # Override the top-level name
# # Override the user-defined sim options, so the simulation
```

```
# # runs forever (until $finish()).
# source ncsim_setup.sh \
# SKIP_FILE_COPY=1 \
# SKIP_DEV_COM=1 \
# SKIP_COM=1 \
# TOP_LEVEL_NAME=top \
# USER_DEFINED_SIM_OPTIONS=""
# # End of template
```

2. Delete the first two characters of each line (comment and space):

```
# Start of template
# If the copied and modified template file is "ncsim.sh", run it as:
# ./ncsim.sh
#
# Do the file copy, dev_com and com steps
source ncsim_setup.sh \
SKIP_ELAB=1 \
SKIP_SIM=1
# Compile the top level module
ncvlog -sv "$QSYS_SIMDIR/../top.sv"
# Do the elaboration and sim steps
# Override the top-level name
# Override the user-defined sim options, so the simulation
# runs forever (until $finish()).
source ncsim_setup.sh \
SKIP_FILE_COPY=1 \
SKIP_DEV_COM=1 \
SKIP_COM=1 \
TOP_LEVEL_NAME=top \
USER_DEFINED_SIM_OPTIONS=""
# End of template
```

3. Modify the TOP_LEVEL_NAME and compilation step appropriately, depending on the simulation's top-level file. For example:

```
TOP_LEVEL_NAME=sim_top \
 ncvlog -sv "$QSYS_SIMDIR/../top.sv"
```

4. If necessary, add the QSYS_SIMDIR variable to point to the location of the generated IP simulation files. Specify any other changes that you require to match your design simulation requirements. The scripts offer variables to set compilation or simulation options. Refer to the generated script for details.
5. Run the resulting top-level script from the generated simulation directory by specifying the path to `ncsim.sh`.

1.6.2.2.3 Sourcing ModelSim* Simulator Setup Scripts

Follow these steps to incorporate the generated ModelSim IP simulation scripts into a top-level project simulation script.

1. The generated simulation script contains the following template lines. Cut and paste these lines into a new file. For example, `sim_top.tcl`.

```
# # Start of template
# # If the copied and modified template file is "mentor.do", run it
# # as: vsim -c -do mentor.do
# #
# # Source the generated sim script
# source msim_setup.tcl
# # Compile eda/sim_lib contents first
# dev_com
# # Override the top-level name (so that elab is useful)
```

```
# set TOP_LEVEL_NAME top
# # Compile the standalone IP.
# com
# # Compile the user top-level
# vlog -sv ../../top.sv
# # Elaborate the design.
# elab
# # Run the simulation
# run -a
# # Report success to the shell
# exit -code 0
# # End of template
```

2. Delete the first two characters of each line (comment and space):

```
# Start of template
# If the copied and modified template file is "mentor.do", run it
# as: vsim -c -do mentor.do
#
# Source the generated sim script source msim_setup.tcl
# Compile eda/sim_lib contents first
dev_com
# Override the top-level name (so that elab is useful)
set TOP_LEVEL_NAME top
# Compile the standalone IP.
com
# Compile the user top-level vlog -sv ../../top.sv
# Elaborate the design.
elab
# Run the simulation
run -a
# Report success to the shell
exit -code 0
# End of template
```

3. Modify the TOP_LEVEL_NAME and compilation step appropriately, depending on the simulation's top-level file. For example:

```
set TOP_LEVEL_NAME sim_top vlog -sv ../../sim_top.sv
```

4. If necessary, add the QSYS_SIMDIR variable to point to the location of the generated IP simulation files. Specify any other changes required to match your design simulation requirements. The scripts offer variables to set compilation or simulation options. Refer to the generated script for details.
5. Run the resulting top-level script from the generated simulation directory:

```
vsim –c –do <path to sim_top>.tcl
```

1.6.2.2.4 Sourcing VCS* Simulator Setup Scripts

Follow these steps to incorporate the generated Synopsys VCS simulation scripts into a top-level project simulation script.

1. The generated simulation script contains these template lines. Cut and paste the lines preceding the "helper file" into a new executable file. For example, synopsys_vcs.f.

```
# # Start of template
# # If the copied and modified template file is "vcs_sim.sh", run it
# # as: ./vcs_sim.sh
# #
# # Override the top-level name
```

```
# # specify a command file containing elaboration options
# # (system verilog extension, and compile the top-level).
# # Override the user-defined sim options, so the simulation
# # runs forever (until $finish()).
# source vcs_setup.sh \
# TOP_LEVEL_NAME=top \
# USER_DEFINED_ELAB_OPTIONS="'-f ../../../synopsys_vcs.f'" \
# USER_DEFINED_SIM_OPTIONS=""
#
# # helper file: synopsys_vcs.f
# +systemverilogext+.sv
# ../../../top.sv
# # End of template
```

2. Delete the first two characters of each line (comment and space) for the `vcs.sh` file, as shown below:

```
# Start of template
# If the copied and modified template file is "vcs_sim.sh", run it
# as: ./vcs_sim.sh
#
# Override the top-level name
# specify a command file containing elaboration options
# (system verilog extension, and compile the top-level).
# Override the user-defined sim options, so the simulation
# runs forever (until $finish()).
source vcs_setup.sh \
TOP_LEVEL_NAME=top \
USER_DEFINED_ELAB_OPTIONS="'-f ../../../synopsys_vcs.f'" \
USER_DEFINED_SIM_OPTIONS=""
```

3. Delete the first two characters of each line (comment and space) for the `synopsys_vcs.f` file, as shown below:

```
# helper file: synopsys_vcs.f
 +systemverilogext+.sv
 ../../../top.sv
# End of template
```

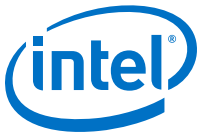
4. Modify the `TOP_LEVEL_NAME` and compilation step appropriately, depending on the simulation's top-level file. For example:

```
TOP_LEVEL_NAME=sim_top \
```

5. If necessary, add the `QSYS_SIMDIR` variable to point to the location of the generated IP simulation files. Specify any other changes required to match your design simulation requirements. The scripts offer variables to set compilation or simulation options. Refer to the generated script for details.
6. Run the resulting top-level script from the generated simulation directory by specifying the path to `vcs_sim.sh`.

1.6.2.2.5 Sourcing VCS* MX Simulator Setup Scripts

Follow these steps to incorporate the generated Synopsys VCS MX simulation scripts for use in top-level project simulation scripts.



1. The generated simulation script contains these template lines. Cut and paste the lines preceding the "helper file" into a new executable file. For example, `vcsmx.sh`.

```
# # Start of template
# # If the copied and modified template file is "vcsmx_sim.sh", run
# # it as: ./vcsmx_sim.sh
# #
# # Do the file copy, dev_com and com steps
# source vcsmx_setup.sh \
# SKIP_ELAB=1 \

# SKIP_SIM=1
#
# # Compile the top level module vlogan +v2k
     +systemverilogext+.sv "$QSYS_SIMDIR/../top.sv"

# # Do the elaboration and sim steps
# # Override the top-level name
# # Override the user-defined sim options, so the simulation runs
# # forever (until $finish()).
# source vcsmx_setup.sh \
# SKIP_FILE_COPY=1 \
# SKIP_DEV_COM=1 \
# SKIP_COM=1 \
# TOP_LEVEL_NAME="'-top top'" \
# USER_DEFINED_SIM_OPTIONS=""
# # End of template
```

2. Delete the first two characters of each line (comment and space), as shown below:

```
# Start of template
# If the copied and modified template file is "vcsmx_sim.sh", run
# it as: ./vcsmx_sim.sh
#
# Do the file copy, dev_com and com steps
source vcsmx_setup.sh \
SKIP_ELAB=1 \
SKIP_SIM=1

# Compile the top level module
vlogan +v2k +systemverilogext+.sv "$QSYS_SIMDIR/../top.sv"

# Do the elaboration and sim steps
# Override the top-level name
# Override the user-defined sim options, so the simulation runs
# forever (until $finish()).
source vcsmx_setup.sh \
SKIP_FILE_COPY=1 \
SKIP_DEV_COM=1 \
SKIP_COM=1 \
```

```
TOP_LEVEL_NAME="'-top top'" \
USER_DEFINED_SIM_OPTIONS=""
# End of template
```

3. Modify the TOP_LEVEL_NAME and compilation step appropriately, depending on the simulation's top-level file. For example:

```
TOP_LEVEL_NAME=”-top sim_top’” \
```

4. Make the appropriate changes to the compilation of the your top-level file, for example:

```
vlogan +v2k +systemverilogext+.sv "$QSYS_SIMDIR/../sim_top.sv"
```

5. If necessary, add the QSYS_SIMDIR variable to point to the location of the generated IP simulation files. Specify any other changes required to match your design simulation requirements. The scripts offer variables to set compilation or simulation options. Refer to the generated script for details.
6. Run the resulting top-level script from the generated simulation directory by specifying the path to `vcsmx_sim.sh`.

1.7 Running a Simulation (Custom Flow)

Use a custom simulation flow to support any of the following more complex simulation scenarios:

- Custom compilation, elaboration, or run commands for your design, IP, or simulation library model files (for example, macros, debugging/optimization options, simulator-specific elaboration or run-time options)
- Multi-pass simulation flows
- Flows that use dynamically generated simulation scripts

Use these to compile libraries and generate simulation scripts for custom simulation flows:

- Simulation Library Compiler—compile Intel FPGA simulation libraries for your device, HDL, and simulator. Generate scripts to compile simulation libraries as part of your custom simulation flow. This tool does not compile your design, IP, or testbench files.
- IP and Qsys Pro simulation scripts—use the scripts generated for Intel FPGA IP cores and Qsys Pro systems as templates to create simulation scripts. If your design includes multiple IP cores or Qsys Pro systems, you can combine the simulation scripts into a single script, manually or by using the `ip-make-simscript` utility.

Use the following steps in a custom simulation flow:

1. Compile the design and testbench files in your simulator.
2. Run the simulation in your simulator.

1.8 Document Revision History

This document has the following revision history.

Date	Version	Changes
2017.05.08	17.0.0	• Updated simulator support table with latest version information.
2016.10.31	16.1.0	• Implemented Intel rebranding. • Removed information about gate-level timing simulation. • Updated simulator support table with latest version information.
2016.05.02	16.0.0	• Removed support for NativeLink in Pro Edition. Added NativeLink support limitations for Standard Edition. • Updated simulator support table with latest version information.
2015.11.02	15.1.0	• Added new Generating Version-Independent IP Simulation Scripts topic. • Added example IP simulation script templates for all supported simulators. • Added new Incorporating IP Simulation Scripts in Top-Level Scripts topic. • Updated simulator support table with latest version information. • Changed instances of *Quartus II* to *Quartus Prime*.
2015.05.04	15.0.0	• Updated simulator support table with latest. • Gate-level timing simulation limited to Stratix IV and Cyclone IV devices. • Added mixed language simulation support in the ModelSim - Intel FPGA Edition software.
2014.06.30	14.0.0	• Replaced MegaWizard Plug-In Manager information with IP Catalog.
May 2013	13.0.0	• Updated introductory section and system and IP file locations.
November 2012	12.1.0	• Revised chapter to reflect latest changes to other simulation documentation.
June 2012	12.0.0	• Reorganization of chapter to reflect various simulation flows. • Added NativeLink support for newer IP cores.
November 2011	11.1.0	• Added information about encrypted Altera simulation model files. • Added information about IP simulation and NativeLink.

Related Links

Altera Documentation Archive

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

2 Timing Analysis Overview

2.1 Timing Analysis Overview

Comprehensive static timing analysis involves analysis of register-to-register, I/O, and asynchronous reset paths. Timing analysis with the TimeQuest Timing Analyzer uses data required times, data arrival times, and clock arrival times to verify circuit performance and detect possible timing violations.

The TimeQuest analyzer determines the timing relationships that must be met for the design to correctly function, and checks arrival times against required times to verify timing. This chapter is an overview of the concepts you need to know to analyze your designs with the TimeQuest analyzer.

Related Links

The Quartus Prime TimeQuest Timing Analyzer on page 41
For more information about the TimeQuest analyzer flow and TimeQuest examples.

2.2 TimeQuest Terminology and Concepts

Table 7. TimeQuest Analyzer Terminology

Term	Definition
nodes	Most basic timing netlist unit. Used to represent ports, pins, and registers.
cells	Look-up tables (LUT), registers, digital signal processing (DSP) blocks, memory blocks, input/output elements, and so on. *Note:* For Intel Stratix® devices, the LUTs and registers are contained in logic elements (LE) and modeled as cells.
pins	Inputs or outputs of cells.
nets	Connections between pins.
ports	Top-level module inputs or outputs; for example, device pins.
clocks	Abstract objects representing clock domains inside or outside of your design.

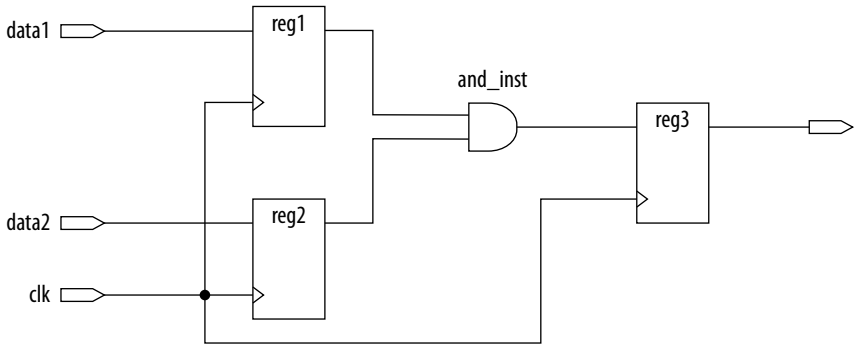
2.2.1 Timing Netlists and Timing Paths

The TimeQuest analyzer requires a timing netlist to perform timing analysis on any design. After you generate a timing netlist, the TimeQuest analyzer uses the data to help determine the different design elements in your design and how to analyze timing.

2.2.1.1 The Timing Netlist

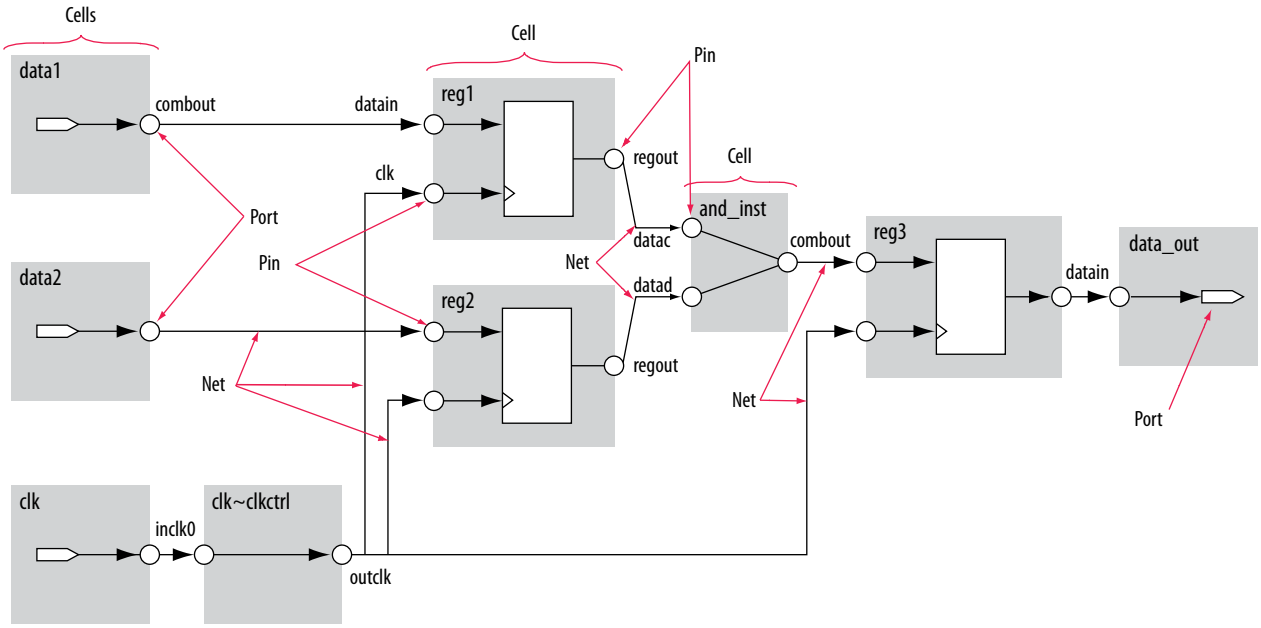
A sample design for which the TimeQuest analyzer generates a timing netlist equivalent.

Figure 2. Sample Design



The timing netlist for the sample design shows how different design elements are divided into cells, pins, nets, and ports.

Figure 3. The TimeQuest Analyzer Timing Netlist



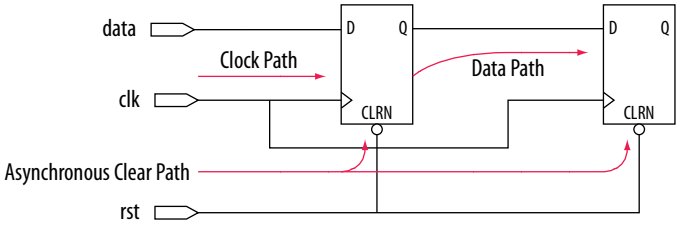
2.2.1.2 Timing Paths

Timing paths connect two design nodes, such as the output of a register to the input of another register.

Understanding the types of timing paths is important to timing closure and optimization. The TimeQuest analyzer uses the following commonly analyzed paths:

- **Edge paths**—connections from ports-to-pins, from pins-to-pins, and from pins-to-ports.
- **Clock paths**—connections from device ports or internally generated clock pins to the clock pin of a register.
- **Data paths**—connections from a port or the data output pin of a sequential element to a port or the data input pin of another sequential element.
- **Asynchronous paths**—connections from a port or asynchronous pins of another sequential element such as an asynchronous reset or asynchronous clear.

Figure 4. Path Types Commonly Analyzed by the TimeQuest Analyzer



In addition to identifying various paths in a design, the TimeQuest analyzer analyzes clock characteristics to compute the worst-case requirement between any two registers in a single register-to-register path. You must constrain all clocks in your design before analyzing clock characteristics.

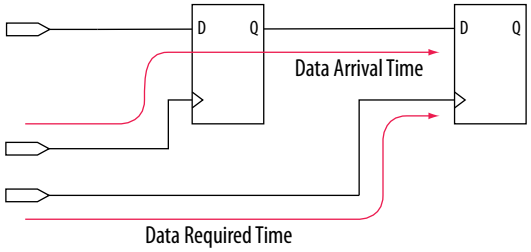
2.2.1.3 Data and Clock Arrival Times

After the TimeQuest analyzer identifies the path type, it can report data and clock arrival times at register pins.

The TimeQuest analyzer calculates data arrival time by adding the launch edge time to the delay from the clock source to the clock pin of the source register, the micro clock-to-output delay (μt_{CO}) of the source register, and the delay from the source register's data output (Q) to the destination register's data input (D).

The TimeQuest analyzer calculates data required time by adding the latch edge time to the sum of all delays between the clock port and the clock pin of the destination register, including any clock port buffer delays, and subtracts the micro setup time (μt_{SU}) of the destination register, where the μt_{SU} is the intrinsic setup time of an internal register in the FPGA.

Figure 5. Data Arrival and Data Required Times



The basic calculations for data arrival and data required times including the launch and latch edges.

Figure 6. Data Arrival and Data Required Time Equations

$$\text{Data Arrival Time} = \text{Launch Edge} + \text{Source Clock Delay} + \mu t_{CO} + \text{Register-to-Register Delay}$$

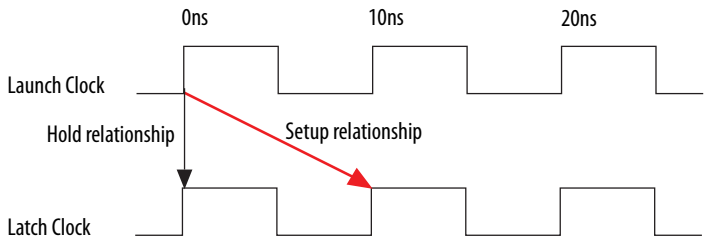
$$\text{Data Required Time} = \text{Latch Edge} + \text{Destination Clock Delay} - \mu t_{SU}$$

2.2.1.4 Launch and Latch Edges

All timing relies on one or more clocks. In addition to analyzing paths, the TimeQuest analyzer determines clock relationships for all register-to-register transfers in your design.

The following figure shows the launch edge, which is the clock edge that sends data out of a register or other sequential element, and acts as a source for the data transfer. A latch edge is the active clock edge that captures data at the data port of a register or other sequential element, acting as a destination for the data transfer. In this example, the launch edge sends the data from register `reg1` at 0 ns, and the register `reg2` captures the data when triggered by the latch edge at 10 ns. The data arrives at the destination register before the next latch edge.

Figure 7. Setup and Hold Relationship for Launch and Latch Edges 10ns Apart



In timing analysis, and with the TimeQuest analyzer specifically, you create clock constraints and assign those constraints to nodes in your design. These clock constraints provide the structure required for repeatable data relationships. The primary relationships between clocks, in the same or different domains, are the setup relationship and the hold relationship.

Note: If you do not constrain the clocks in your design, the Quartus Prime software analyzes in terms of a 1 GHz clock to maximize timing based Fitter effort. To ensure realistic slack values, you must constrain all clocks in your design with real values.

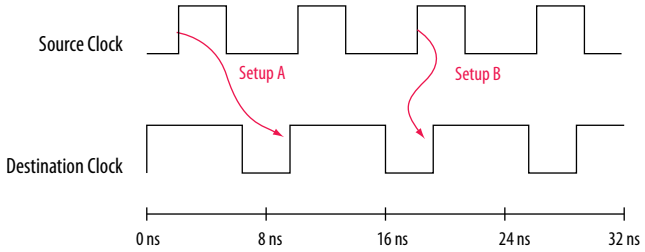
2.2.2 Clock Setup Check

To perform a clock setup check, the TimeQuest analyzer determines a setup relationship by analyzing each launch and latch edge for each register-to-register path.

For each latch edge at the destination register, the TimeQuest analyzer uses the closest previous clock edge at the source register as the launch edge. The following figure shows two setup relationships, setup A and setup B. For the latch edge at 10 ns, the closest clock that acts as a launch edge is at 3 ns and is labeled setup A. For the latch edge at 20 ns, the closest clock that acts as a launch edge is 19 ns and is

labeled setup B. TimQuest analyzes the most restrictive setup relationship, in this case setup B; if that relationship meets the design requirement, then setup A meets it by default.

Figure 8. Setup Check



The TimeQuest analyzer reports the result of clock setup checks as slack values. Slack is the margin by which a timing requirement is met or not met. Positive slack indicates the margin by which a requirement is met; negative slack indicates the margin by which a requirement is not met.

Figure 9. Clock Setup Slack for Internal Register-to-Register Paths

Clock Setup Slack	=	Data Required Time – Data Arrival Time
Data Arrival Time	=	Launch Edge + Clock Network Delay to Source Register + μt_{CO} + Register-to-Register Delay
Data Required Time	=	Latch Edge + Clock Network Delay to Destination Register – μt_{SU} – Setup Uncertainty

The TimeQuest analyzer performs setup checks using the maximum delay when calculating data arrival time, and minimum delay when calculating data required time.

Figure 10. Clock Setup Slack from Input Port to Internal Register

Clock Setup Slack	=	Data Required Time – Data Arrival Time
Data Arrival Time	=	Launch Edge + Clock Network Delay + Input Maximum Delay + Port-to-Register Delay
Data Required Time	=	Latch Edge + Clock Network Delay to Destination Register – μt_{SU} – Setup Uncertainty

Figure 11. Clock Setup Slack from Internal Register to Output Port

Clock Setup Slack	=	Data Required Time – Data Arrival Time
Data Required Time	=	Latch Edge + Clock Network Delay to Output Port – Output Maximum Delay
Data Arrival Time	=	Launch Edge + Clock Network Delay to Source Register +μt_{CO} + Register-to-Port Delay

2.2.3 Clock Hold Check

To perform a clock hold check, the TimeQuest analyzer determines a hold relationship for each possible setup relationship that exists for all source and destination register pairs. The TimeQuest analyzer checks all adjacent clock edges from all setup relationships to determine the hold relationships.

The TimeQuest analyzer performs two hold checks for each setup relationship. The first hold check determines that the data launched by the current launch edge is not captured by the previous latch edge. The second hold check determines that the data launched by the next launch edge is not captured by the current latch edge. From the possible hold relationships, the TimeQuest analyzer selects the hold relationship that is the most restrictive. The most restrictive hold relationship is the hold relationship

with the smallest difference between the latch and launch edges and determines the minimum allowable delay for the register-to-register path. In the following example, the TimeQuest analyzer selects hold check A2 as the most restrictive hold relationship of two setup relationships, setup A and setup B, and their respective hold checks.

Figure 12. Setup and Hold Check Relationships

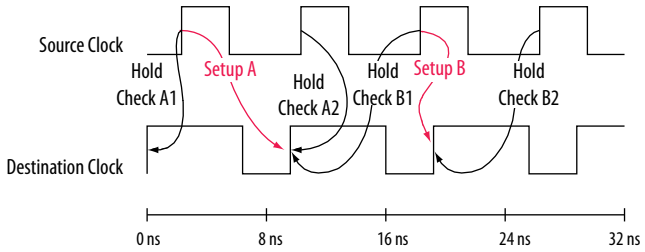


Figure 13. Clock Hold Slack for Internal Register-to-Register Paths

$$\text{Clock Hold Slack} = \text{Data Arrival Time} - \text{Data Required Time}$$
$$\text{Data Arrival Time} = \text{Launch Edge} + \text{Clock Network Delay to Source Register} + \mu t_{CO} + \text{Register-to-Register Delay}$$
$$\text{Data Required Time} = \text{Latch Edge} + \text{Clock Network Delay to Destination Register} + \mu t_{H} + \text{Hold Uncertainty}$$

The TimeQuest analyzer performs hold checks using the minimum delay when calculating data arrival time, and maximum delay when calculating data required time.

Figure 14. Clock Hold Slack Calculation from Input Port to Internal Register

$$\text{Clock Hold Slack} = \text{Data Arrival Time} - \text{Data Required Time}$$
$$\text{Data Arrival Time} = \text{Launch Edge} + \text{Clock Network Delay} + \text{Input Minimum Delay} + \text{Pin-to-Register Delay}$$
$$\text{Data Required Time} = \text{Latch Edge} + \text{Clock Network Delay to Destination Register} + \mu t_{H}$$

Figure 15. Clock Hold Slack Calculation from Internal Register to Output Port

$$\text{Clock Hold Slack} = \text{Data Arrival Time} - \text{Data Required Time}$$
$$\text{Data Arrival Time} = \text{Launch Edge} + \text{Clock Network Delay to Source Register} + \mu t_{CO} + \text{Register-to-Pin Delay}$$
$$\text{Data Required Time} = \text{Latch Edge} + \text{Clock Network Delay} - \text{Output Minimum Delay}$$

2.2.4 Recovery and Removal Time

Recovery time is the minimum length of time for the deassertion of an asynchronous control signal relative to the next clock edge.

For example, signals such as `clear` and `preset` must be stable before the next active clock edge. The recovery slack calculation is similar to the clock setup slack calculation, but it applies to asynchronous control signals.

Figure 16. Recovery Slack Calculation if the Asynchronous Control Signal is Registered

$$\text{Recovery Slack Time} = \text{Data Required Time} - \text{Data Arrival Time}$$
$$\text{Data Required Time} = \text{Latch Edge} + \text{Clock Network Delay to Destination Register} - \mu t_{SU}$$
$$\text{Data Arrival Time} = \text{Launch Edge} + \text{Clock Network Delay to Source Register} + \mu t_{CO} + \text{Register-to-Register Delay}$$

Figure 17. Recovery Slack Calculation if the Asynchronous Control Signal is not Registered

Recovery Slack Time = Data Required Time – Data Arrival Time
Data Required Time = Latch Edge + Clock Network Delay to Destination Register – μt_{SU}
Data Arrival Time = Launch Edge + Clock Network Delay + Input Maximum Delay + Port-to-Register Delay

Note: If the asynchronous reset signal is from a device I/O port, you must create an input delay constraint for the asynchronous reset port for the TimeQuest analyzer to perform recovery analysis on the path.

Removal time is the minimum length of time the deassertion of an asynchronous control signal must be stable after the active clock edge. The TimeQuest analyzer removal slack calculation is similar to the clock hold slack calculation, but it applies asynchronous control signals.

Figure 18. Removal Slack Calcuation if the Asynchronous Control Signal is Registered

Removal Slack Time = Data Arrival Time – Data Required Time
Data Arrival Time = Launch Edge + Clock Network Delay to Source Register + μt_{CO} of Source Register + Register-to-Register Delay
Data Required Time = Latch Edge + Clock Network Delay to Destination Register + μt_{H}

Figure 19. Removal Slack Calculation if the Asynchronous Control Signal is not Registered

Removal Slack Time = Data Arrival Time – Data Required Time
Data Arrival Time = Launch Edge + Clock Network Delay + Input Minimum Delay of Pin + Minimum Pin-to-Register Delay
Data Required Time = Latch Edge + Clock Network Delay to Destination Register + μt_{H}

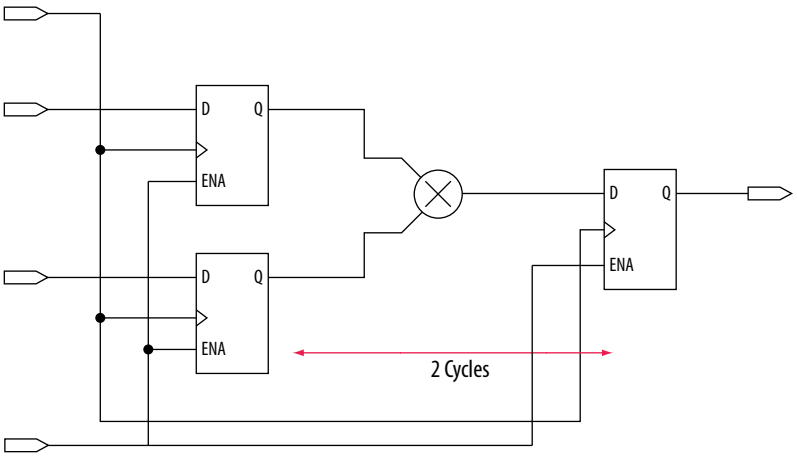
If the asynchronous reset signal is from a device pin, you must assign the **Input Minimum Delay** timing assignment to the asynchronous reset pin for the TimeQuest analyzer to perform removal analysis on the path.

2.2.5 Multicycle Paths

Multicycle paths are data paths that require a non-default setup and/or hold relationship for proper analysis.

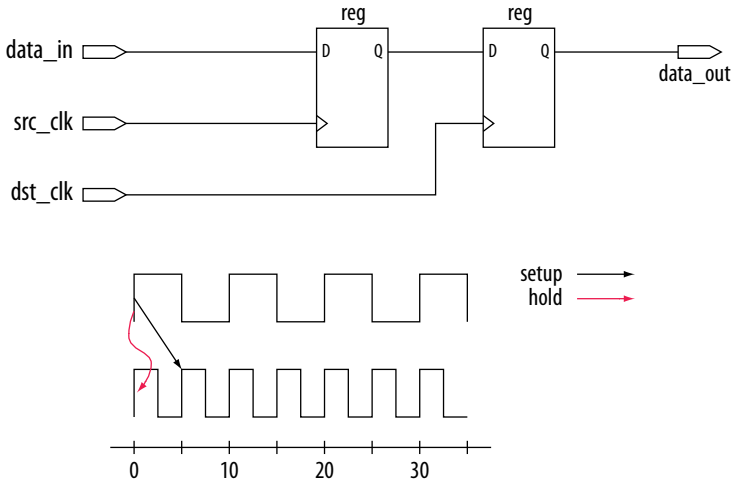
For example, a register may be required to capture data on every second or third rising clock edge. An example of a multicycle path between the input registers of a multiplier and an output register where the destination latches data on every other clock edge.

Figure 20. Multicycle Path



A register-to-register path used for the default setup and hold relationship, the respective timing diagrams for the source and destination clocks, and the default setup and hold relationships, when the source clock, `src_clk`, has a period of 10 ns and the destination clock, `dst_clk`, has a period of 5 ns. The default setup relationship is 5 ns; the default hold relationship is 0 ns.

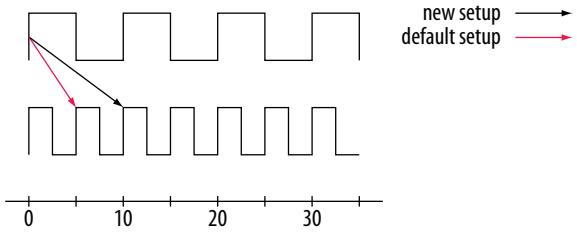
Figure 21. Register-to-Register Path and Default Setup and Hold Timing Diagram



To accommodate the system requirements you can modify the default setup and hold relationships with a multicycle timing exception.

The actual setup relationship after you apply a multicycle timing exception. The exception has a multicycle setup assignment of two to use the second occurring latch edge; in this example, to 10 ns from the default value of 5 ns.

Figure 22. Modified Setup Diagram



Related Links

The Quartus Prime TimeQuest Timing Analyzer on page 41
For more information about creating exceptions with multicycle paths.

2.2.6 Metastability

Metastability problems can occur when a signal is transferred between circuitry in unrelated or asynchronous clock domains because the designer cannot guarantee that the signal will meet setup and hold time requirements.

To minimize the failures due to metastability, circuit designers typically use a sequence of registers, also known as a synchronization register chain, or synchronizer, in the destination clock domain to resynchronize the data signals to the new clock domain.

The mean time between failures (MTBF) is an estimate of the average time between instances of failure due to metastability.

The TimeQuest analyzer analyzes the potential for metastability in your design and can calculate the MTBF for synchronization register chains. The MTBF of the entire design is then estimated based on the synchronization chains it contains.

In addition to reporting synchronization register chains found in the design, the Quartus Prime software also protects these registers from optimizations that might negatively impact MTBF, such as register duplication and logic retiming. The Quartus Prime software can also optimize the MTBF of your design if the MTBF is too low.

Related Links

- Understanding Metastability in FPGAs
  For more information about metastability, its effects in FPGAs, and how MTBF is calculated.
- Managing Metastability with the Quartus Prime Software
  For more information about metastability analysis, reporting, and optimization features in the Quartus Prime software.

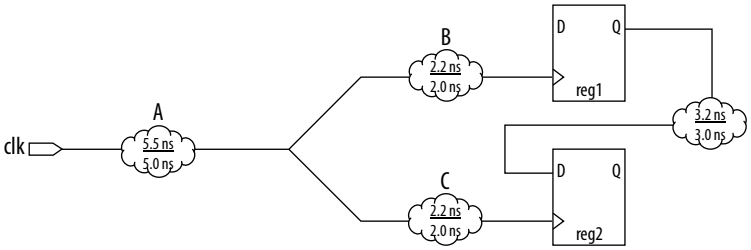
2.2.7 Common Clock Path Pessimism Removal

Common clock path pessimism removal accounts for the minimum and maximum delay variation associated with common clock paths during static timing analysis by adding the difference between the maximum and minimum delay value of the common clock path to the appropriate slack equation.

Minimum and maximum delay variation can occur when two different delay values are used for the same clock path. For example, in a simple setup analysis, the maximum clock path delay to the source register is used to determine the data arrival time. The

minimum clock path delay to the destination register is used to determine the data required time. However, if the clock path to the source register and to the destination register share a common clock path, both the maximum delay and the minimum delay are used to model the common clock path during timing analysis. The use of both the minimum delay and maximum delay results in an overly pessimistic analysis since two different delay values, the maximum and minimum delays, cannot be used to model the same clock path.

Figure 23. Typical Register to Register Path

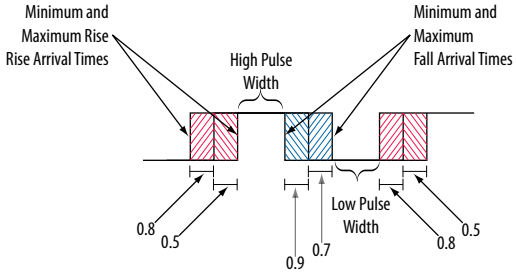


Segment A is the common clock path between `reg1` and `reg2`. The minimum delay is 5.0 ns; the maximum delay is 5.5 ns. The difference between the maximum and minimum delay value equals the common clock path pessimism removal value; in this case, the common clock path pessimism is 0.5 ns. The TimeQuest analyzer adds the common clock path pessimism removal value to the appropriate slack equation to determine overall slack. Therefore, if the setup slack for the register-to-register path in the example equals 0.7 ns without common clock path pessimism removal, the slack would be 1.2 ns with common clock path pessimism removal.

You can also use common clock path pessimism removal to determine the minimum pulse width of a register. A clock signal must meet a register's minimum pulse width requirement to be recognized by the register. A minimum high time defines the minimum pulse width for a positive-edge triggered register. A minimum low time defines the minimum pulse width for a negative-edge triggered register.

Clock pulses that violate the minimum pulse width of a register prevent data from being latched at the data pin of the register. To calculate the slack of the minimum pulse width, the TimeQuest analyzer subtracts the required minimum pulse width time from the actual minimum pulse width time. The TimeQuest analyzer determines the actual minimum pulse width time by the clock requirement you specified for the clock that feeds the clock port of the register. The TimeQuest analyzer determines the required minimum pulse width time by the maximum rise, minimum rise, maximum fall, and minimum fall times.

Figure 24. Required Minimum Pulse Width time for the High and Low Pulse



With common clock path pessimism, the minimum pulse width slack can be increased by the smallest value of either the maximum rise time minus the minimum rise time, or the maximum fall time minus the minimum fall time. In the example, the slack value can be increased by 0.2 ns, which is the smallest value between 0.3 ns (0.8 ns – 0.5 ns) and 0.2 ns (0.9 ns – 0.7 ns).

Related Links

TimeQuest Timing Analyzer Page (Settings Dialog Box)
For more information, refer to the Quartus Prime Help.

2.2.8 Clock-As-Data Analysis

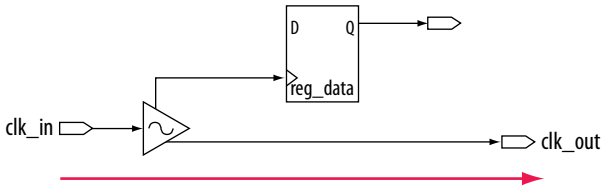
The majority of FPGA designs contain simple connections between any two nodes known as either a data path or a clock path.

A data path is a connection between the output of a synchronous element to the input of another synchronous element.

A clock is a connection to the clock pin of a synchronous element. However, for more complex FPGA designs, such as designs that use source-synchronous interfaces, this simplified view is no longer sufficient. Clock-as-data analysis is performed in circuits with elements such as clock dividers and DDR source-synchronous outputs.

The connection between the input clock port and output clock port can be treated either as a clock path or a data path. A design where the path from port `clk_in` to port `clk_out` is both a clock and a data path. The clock path is from the port `clk_in` to the register `reg_data` clock pin. The data path is from port `clk_in` to the port `clk_out`.

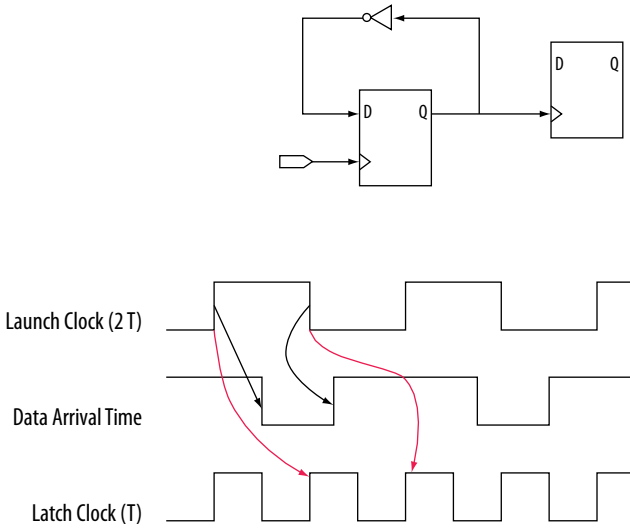
Figure 25. Simplified Source Synchronous Output



With clock-as-data analysis, the TimeQuest analyzer provides a more accurate analysis of the path based on user constraints. For the clock path analysis, any phase shift associated with the phase-locked loop (PLL) is taken into consideration. For the data path analysis, any phase shift associated with the PLL is taken into consideration rather than ignored.

The clock-as-data analysis also applies to internally generated clock dividers. An internally generated clock divider. In this figure, waveforms are for the inverter feedback path, analyzed during timing analysis. The output of the divider register is used to determine the launch time and the clock port of the register is used to determine the latch time.

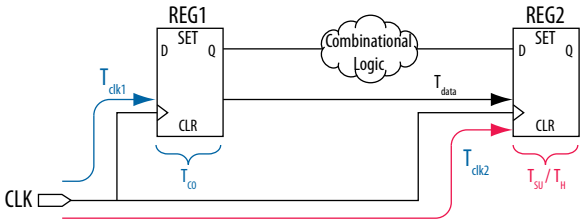
Figure 26. Clock Divider



2.2.9 Multicycle Clock Setup Check and Hold Check Analysis

You can modify the setup and hold relationship when you apply a multicycle exception to a register-to-register path.

Figure 27. Register-to-Register Path

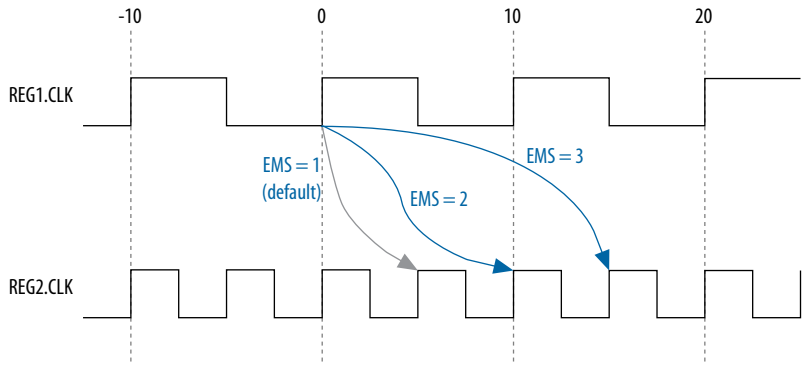


2.2.9.1 Multicycle Clock Setup

The setup relationship is defined as the number of clock periods between the latch edge and the launch edge. By default, the TimeQuest analyzer performs a single-cycle path analysis, which results in the setup relationship being equal to one clock period (latch edge – launch edge). Applying a multicycle setup assignment, adjusts the setup relationship by the multicycle setup value. The adjustment value may be negative.

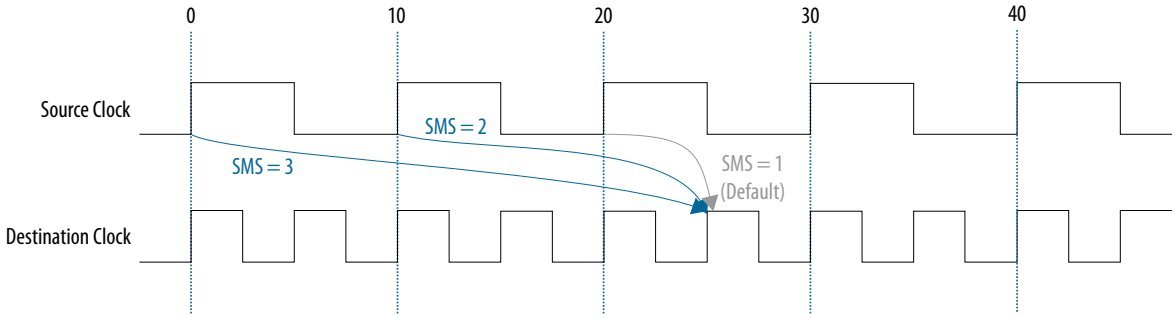
An end multicycle setup assignment modifies the latch edge of the destination clock by moving the latch edge the specified number of clock periods to the right of the determined default latch edge. The following figure shows various values of the end multicycle setup (EMS) assignment and the resulting latch edge.

Figure 28. End Multicycle Setup Values



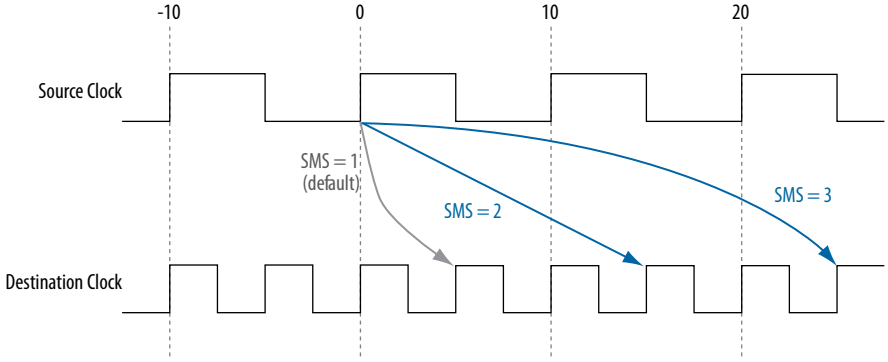
A start multicycle setup assignment modifies the launch edge of the source clock by moving the launch edge the specified number of clock periods to the left of the determined default launch edge. A start multicycle setup (SMS) assignment with various values can result in a specific launch edge.

Figure 29. Start Multicycle Setup Values



The setup relationship reported by the TimeQuest analyzer for the negative setup relationship.

Figure 30. Start Multicycle Setup Values Reported by the TimeQuest Analyzer



2.2.9.2 Multicycle Clock Hold

The setup relationship is defined as the number of clock periods between the launch edge and the latch edge.

By default, the TimeQuest analyzer performs a single-cycle path analysis, which results in the hold relationship being equal to one clock period (launch edge – latch edge).When analyzing a path, the TimeQuest analyzer performs two hold checks. The first hold check determines that the data launched by the current launch edge is not captured by the previous latch edge. The second hold check determines that the data launched by the next launch edge is not captured by the current latch edge. The TimeQuest analyzer reports only the most restrictive hold check. The TimeQuest analyzer calculates the hold check by comparing launch and latch edges.

The calculation the TimeQuest analyzer performs to determine the hold check.

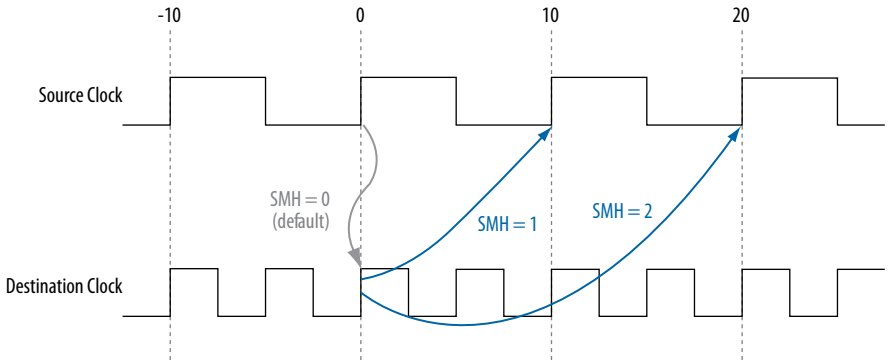
Figure 31. Hold Check

$$\text{hold check 1} = \text{current launch edge} - \text{previous latch edge}$$
$$\text{hold check 2} = \text{next launch edge} - \text{current latch edge}$$

Tip: If a hold check overlaps a setup check, the hold check is ignored.

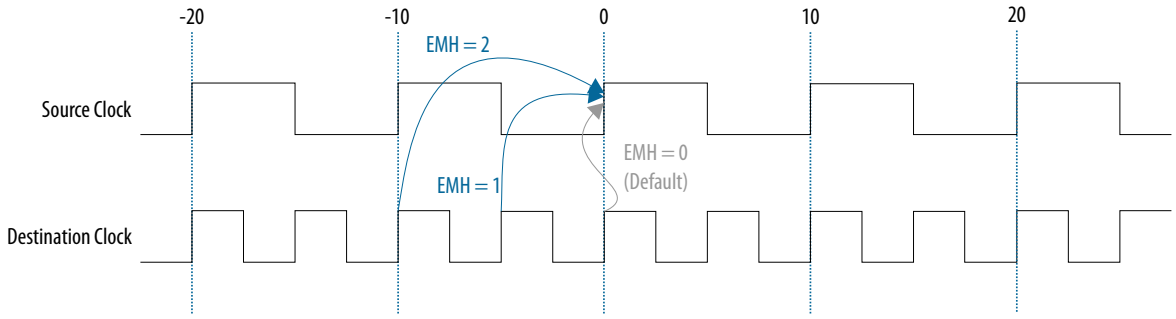
A start multicycle hold assignment modifies the launch edge of the destination clock by moving the latch edge the specified number of clock periods to the right of the determined default launch edge. The following figure shows various values of the start multicycle hold (SMH) assignment and the resulting launch edge.

Figure 32. Start Multicycle Hold Values



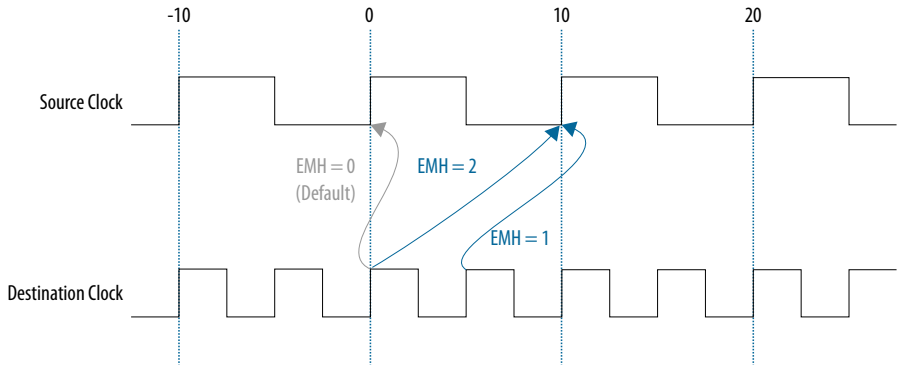
An end multicycle hold assignment modifies the latch edge of the destination clock by moving the latch edge the specific ed number of clock periods to the left of the determined default latch edge. The following figure shows various values of the end multicycle hold (EMH) assignment and the resulting latch edge.

Figure 33. End Multicycle Hold Values



The hold relationship reported by the TimeQuest analyzer for the negative hold relationship shown in the figure above would look like this:

Figure 34. End Multicycle Hold Values Reported by the TimeQuest Analyzer



2.2.10 Multicorner Analysis

The TimeQuest analyzer performs multicorner timing analysis to verify your design under a variety of operating conditions—such as voltage, process, and temperature—while performing static timing analysis.

To change the operating conditions or speed grade of the device used for timing analysis, use the `set_operating_conditions` command.

If you specify an operating condition Tcl object, the `-model`, `speed`, `-temperature`, and `-voltage` options are optional. If you do not specify an operating condition Tcl object, the `-model` option is required; the `-speed`, `-temperature`, and `-voltage` options are optional.

Tip: To obtain a list of available operating conditions for the target device, use the `get_available_operating_conditions -all` command.

To ensure that no violations occur under various conditions during the device operation, perform static timing analysis under all available operating conditions.

Table 8. Operating Conditions for Slow and Fast Models

Model	Speed Grade	Voltage	Temperature
Slow	Slowest speed grade in device density	V_{cc} minimum supply (1)	Maximum T_J (1)
Fast	Fastest speed grade in device density	V_{cc} maximum supply (1)	Minimum T_J (1)

Note :
1. Refer to the DC & Switching Characteristics chapter of the applicable device Handbook for V_{cc} and T_J. values

In your design, you can set the operating conditions for to the slow timing model, with a voltage of 1100 mV, and temperature of 85° C with the following code:

```
set_operating_conditions -model slow -temperature 85 -voltage 1100
```

You can set the same operating conditions with a Tcl object:

```
set_operating_conditions 3_slow_1100mv_85c
```

The following block of code shows how to use the `set_operating_conditions` command to generate different reports for various operating conditions.

Example 1. Script Excerpt for Analysis of Various Operating Conditions

```
#Specify initial operating conditions
set_operating_conditions -model slow -speed 3 -grade c -temperature 85 -voltage
1100
#Update the timing netlist with the initial conditions
update_timing_netlist
#Perform reporting
#Change initial operating conditions. Use a temperature of 0C
set_operating_conditions -model slow -speed 3 -grade c -temperature 0 -voltage
1100
#Update the timing netlist with the new operating condition
update_timing_netlist
#Perform reporting
#Change initial operating conditions. Use a temperature of 0C and a model of fast
set_operating_conditions -model fast -speed 3 -grade c -temperature 0 -voltage
1100
#Update the timing netlist with the new operating condition
update_timing_netlist
#Perform reporting
```

Related Links

- set_operating_conditions
  For more information about the `set_operating_conditions` command
- get_available_operating_conditions
  For more information about the `get_available_operating_conditions` command

2.3 Document Revision History

Table 9. Document Revision History

Date	Version	Changes
2016.10.31	16.1.0	• Implemented Intel rebranding.
2016.05.02	16.0.0	Corrected typo in Fig 6-14: Clock Hold Slack Calculation from Internal Register to Output Port
2015.11.02	15.1.0	Changed instances of *Quartus II* to *Quartus Prime*.
2014.12.15	14.1.0	Moved Multicycle Clock Setup Check and Hold Check Analysis section from the TimeQuest Timing Analyzer chapter.
June 2014	14.0.0	Updated format
June 2012	12.0.0	Added social networking icons, minor text updates
November 2011	11.1.0	Initial release.

Related Links

Altera Documentation Archive

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

3 The Quartus Prime TimeQuest Timing Analyzer

The Quartus Prime TimeQuest Timing Analyzer is a powerful ASIC-style timing analysis tool that validates the timing performance of all logic in your design using an industry-standard constraint, analysis, and reporting methodology. Use the TimeQuest analyzer GUI or command-line interface to constrain, analyze, and report results for all timing paths in your design.

This document is organized to allow you to refer to specific subjects relating to the TimeQuest analyzer and timing analysis. The sections cover the following topics:

Related Links

- Timing Analysis Overview on page 24
- TimeQuest Timing Analyzer Resource Center
- Intel FPGA Technical Training

3.1 Enhanced Timing Analysis for Intel® Arria® 10 Devices

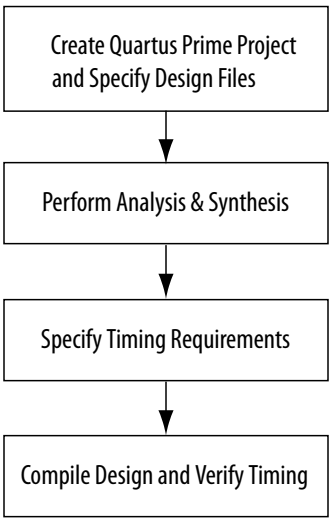
The TimeQuest Timing Analyzer supports new timing algorithms for the Arria® 10 device family which significantly improve the speed of the analysis.

These algorithms are enabled by default for Arria 10 devices, and can be enabled for earlier families with an assignment. The new analysis engine analyzes the timing graph a fixed number of times. Previous TimeQuest analysis analyzed the timing graph as many times as there were constraints in your Synopsys Design Constraint (SDC) file.

3.2 Recommended Flow for First Time Users

The Quartus Prime TimeQuest analyzer performs constraint validation to timing verification as part of the compilation flow. Both the TimeQuest analyzer and the Fitter use of constraints contained in a Synopsys Design Constraints (`.sdc`) file. The following flow is recommended if you have not created a project and do not have a SDC file with timing constraints for your design.

Figure 35. Design Flow with the TimeQuest Timing Analyzer



3.2.1 Creating and Setting Up your Design

You must first create your project in the Quartus Prime software. Include all the necessary design files, including any existing Synopsys Design Constraints (`.sdc`) files, also referred to as SDC files, that contain timing constraints for your design. Some reference designs, or Intel FPGA or partner IP cores may already include one or more SDC files.

All SDC files must be added to your project so that your constraints are processed when the Quartus Prime software performs Fitting and Timing Analysis. Typically you must create an SDC file to constrain your design.

Related Links

SDC File Precedence on page 100
The Fitter and the TimeQuest analyzer process SDC files in the order you specify in the Quartus Prime Settings File (`.qsf`).

3.2.2 Specifying Timing Requirements

Before running timing analysis with the TimeQuest analyzer, you must specify timing constraints, describe the clock frequency requirements and other characteristics, timing exceptions, and I/O timing requirements of your design. You can use the TimeQuest Timing Analyzer Wizard to enter initial constraints for your design, and then refine timing constraints with the TimeQuest analyzer GUI.

Both the TimeQuest analyzer and the Fitter use of constraints contained in a Synopsis Design Constraints (`.sdc`) file.

The constraints in the SDC file are read in sequence. You must first make a constraint before making any references to that constraint. For example, if a generated clock references a base clock, the base clock constraint must be made before the generated clock constraint.

If you are new to timing analysis with the TimeQuest analyzer, you can use template files included with the Quartus Prime software and the interactive dialog boxes to create your initial SDC file. To use this method, refer to Performing an Initial Analysis and Synthesis.

If you are familiar with timing analysis, you can also create an SDC file in you preferred text editor. Don't forget to include the SDC file in the project when you are finished.

3.2.2.1 Performing an Initial Analysis and Synthesis

Perform Analysis and Synthesis on your design so that you can find design entry names in the **Node Finder** to simplify creating constraints.

The Quartus Prime software populates an internal database with design element names. You must synthesize your design in order for the Quartus Prime software to assign names to your design elements, for example, pins, nodes, hierarchies, and timing paths.

If you have already compiled your design, you do not need need to perform the synthesis step again, because compiling the design automatically performs synthesis.You can either perform Analysis and Synthesis to create a post-map database, or perform a full compilation to create a post-fit database. Creating a post-map database is faster than a post-fit database, and is sufficient for creating initial timing constraints.

Note: When compiling for the Arria 10 device family, the following commands are required to perform initial synthesis and enable you to use the **Node Finder** to find names in your design:

```
quartus_map <design>
quartus_fit <design> --floorplan
quartus_sta <design> --post_map
```

When compiling for other devices, you can exclude the `quartus_fit <design> --floorplan` step:

```
quartus_map <design>
quartus_sta <design> --post_map
```

3.2.2.2 Creating a Constraint File from Quartus Prime Templates with the Quartus Prime Text Editor

You can create an SDC file from constraint templates in the Quartus Prime software with the Quartus Prime Text Editor, or with your preferred text editor.

1. On the **File** menu, click **New**.
2. In the **New** dialog box, select the **Synopsys Design Constraints File** type from the **Other Files** group. Click **OK**.
3. Right-click in the blank SDC file in the Quartus Prime Text Editor, then click **Insert Constraint**. Choose **Clock Constraint** followed by **Set Clock Groups** since they are the most widely used constraints.
   The Quartus Prime Text Editor displays a dialog box with interactive fields for creating constraints. For example, the **Create Clock** dialog box shows you the waveform for your `create_clock` constraint while you adjust the **Period** and **Rising** and **Falling** waveform edge settings. The actual constraint is displayed in the **SDC command** field. Click **Insert** to use the constraint in your SDC.

   or
4. Click the **Insert Template** button on the text editor menu, or, right-click in the blank SDC file in the Quartus Prime Text Editor, then click **Insert Template**TimeQuest.
   a. In the **Insert Template** dialog box, expand the **TimeQuest** section, then expand the **SDC Commands** section.
   b. Expand a command category, for example, **Clocks**.
   c. Select a command. The SDC constraint appears in the **Preview** pane.
   d. Click **Insert** to paste the SDC constraint into the blank SDC file you created in step 2.
      This creates a generic constraint for you to edit manually, replacing variables such as clock names, period, rising and falling edges, ports, etc.
5. Repeat as needed with other constraints, or click **Close** to close the **Insert Template** dialog box.

You can now use any of the standard features of the Quartus Prime Text Editor to modify the SDC file, or save the SDC file to edit in a text editor. Your SDC can be saved with the same name as the project, and generally should be stored in the project directory.

Related Links

- Create Clocks Dialog Box
- Set Clock Groups Dialog Box
  For more information on Create Clocks and Set Clock Groups, refer to the Quartus Prime Help.

3.2.3 Performing a Full Compilation

After creating initial timing constraints, compile your design.

During a full compilation, the Fitter uses the TimeQuest analyzer repeatedly to perform timing analysis with your timing constraints. By default, the Fitter can stop early if it meets your timing requirements, instead of attempting to achieve the maximum performance. You can modify this by changing the Fitter effort settings in the Quartus Prime software.

If you are using the Quartus Prime Pro Edition software, you have some command line options available when performing iterative timing analysis on large designs.You can perform a less intensive analysis with `quartus_sta --mode=implement`. In this mode, the Quartus Prime software performs a reduced-corner timing analysis. When your modification iterations have achieved the desired result, you can use `quartus_sta --mode=finalize` to perform final Fitter optimizations and a full four-corner timing analysis.

Related Links

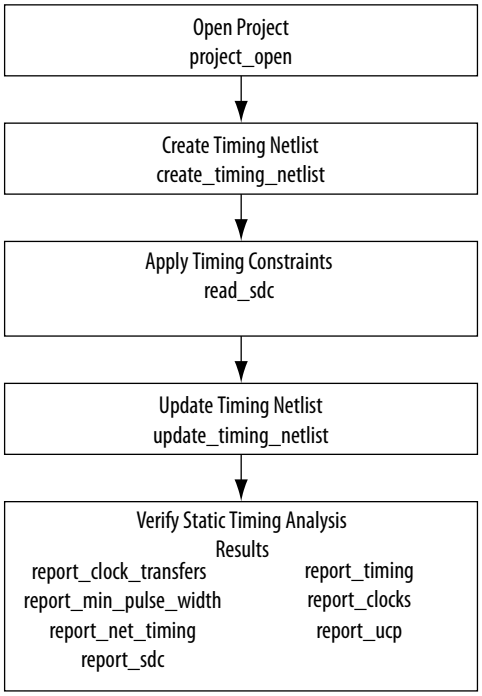
- Analyzing Timing in Designs Compiled in Previous Versions on page 46
  For more information about importing databases compiled in previous versions of the software.
- Fitter Settings Page (Settings Dialog Box)
  For more information about changing Fitter effort, refer to the Quartus Prime Help.

3.2.4 Verifying Timing

The TimeQuest analyzer examines the timing paths in the design, calculates the propagation delay along each path, checks for timing constraint violations, and reports timing results as positive slack or negative slack. Negative slack indicates a timing violation. If you encounter violations along timing paths, use the timing reports to analyze your design and determine how best to optimize your design. If you modify, remove, or add constraints, you should perform a full compilation again. This iterative process helps resolve timing violations in your design.

There is a recommended flow for constraining and analyzing your design within the TimeQuest analyzer, and each part of the flow has a corresponding Tcl command.

Figure 36. The TimeQuest Timing Analyzer Flow



3.2.5 Analyzing Timing in Designs Compiled in Previous Versions

Performing a full compilation can be a lengthy process, however, once your design meets timing you can export the design database for later use. This can include operations such as verification of subsequent timing models, or use in a later version of the Quartus Prime software.

When you re-open the project, the Quartus Prime software opens the exported database from the export directory. You can then run TimeQuest on the design without having to recompile the project.

To export the database in the previous version of the Quartus Prime software, click **Project ➤ Export Database** and select the export directory to contain the exported database.

To import a database in a later version of the Quartus Prime software, click **File ➤ Open** and select the Quartus Prime Project file (`.qpf`) for the project.

Once you import the database, you can perform any TimeQuest analyzer functions on the design without recompiling.

3.3 Timing Constraints

Timing analysis in the Quartus Prime software with the TimeQuest Timing Analyzer relies on constraining your design to make it meet your timing requirements. When discussing these constraints, they can be referred to as timing constraints, SDC constraints, or SDC commands interchangeably.

3.3.1 Recommended Starting SDC Constraints

Almost every beginning SDC file should contain the following four commands:

create_clock on page 47

derive_pll_clocks on page 48

derive_clock_uncertainty on page 49

SDC Constraint Creation Summary on page 49

set_clock_groups on page 49

Related Links

Creating a Constraint File from Quartus Prime Templates with the Quartus Prime Text Editor on page 44

You can create an SDC file from constraint templates in the Quartus Prime software with the Quartus Prime Text Editor, or with your preferred text editor.

3.3.1.1 create_clock

The first statements in a SDC file should be constraints for clocks, for example, constrain the external clocks coming into the FPGA with `create_clock`. An example of the basic syntax is:

```
create_clock -name sys_clk -period 8.0 [get_ports fpga_clk]
```

This command creates a clock called `sys_clk` with an 8ns period and applies it to the port called `fpga_clk`.

Note: Both Tcl files and SDC files are case-sensitive, so make sure references to pins, ports, or nodes, such as `fpga_clk` match the case used in your design.

By default, the clock has a rising edge at time 0ns, and a 50% duty cycle, hence a falling edge at time 4ns. If you require a different duty cycle or to represent an offset, use the `-waveform` option, however, this is seldom necessary.

It is common to create a clock with the same name as the port it is applied to. In the example above, this would be accomplished by:

```
create_clock -name fpga_clk -period 8.0 [get_ports fpga_clk]
```

There are now two unique objects called `fpga_clk`, a port in your design and a clock applied to that port.

Note: In Tcl syntax, square brackets execute the command inside them, so `[get_ports fpga_clk]` executes a command that finds all ports in the design that match `fpga_clk` and returns a collection of them. You can enter the command without using the `get_ports` collection command, as shown in the following example. There are benefits to using collection commands, which are described in "Collection Commands".

```
create_clock -name sys_clk -period 8.0 fpga_clk
```

Repeat this process, using one create_clock command for each known clock coming into your design. Later on you can use **Report Unconstrained Paths** to identify any unconstrained clocks.

Note: Rather than typing constraints, users can enter constraints through the GUI. After launching TimeQuest, open the SDC file from TimeQuest or Quartus Prime, place the cursor where the new constraint will go, and go to **Edit ➤ Insert Constraint**, and choose the constraint.

Warning: Using the **Constraints** menu option in the TimeQuest GUI applies constraints directly to the timing database, but makes no entry in the SDC file. An advanced user may find reasons to do this, but if you are new to TimeQuest, Intel FPGA recommends entering your constraints directly into your SDC with the **Edit ➤ Insert Constraint** command.

Related Links

Creating Base Clocks on page 53
Base clocks are the primary input clocks to the device.

3.3.1.2 derive_pll_clocks

After the `create_clock` commands add the following command into your SDC file:

```
derive_pll_clocks
```

This command automatically creates a generated clock constraint on each output of the PLLs in your design..

When PLLs are created, you define how each PLL output is configured. Because of this, the TimeQuest analyzer can automatically constrain them, wit thet `derive_pll_clocks` command.

This command also creates other constraints. It constrains transceiver clocks. It adds multicycles between LVDS SERDES and user logic.

The **derive_pll_clocks** command prints an Info message to show each generated clock it creates.

If you are new to the TimeQuest analyzer, you may decide not to use `derive_pll_clocks`, and instead cut-and-paste each `create_generated_clock` assignment into the SDC file. There is nothing wrong with this, since the two are identical. The problem is that when you modifiy a PLL setting, you must remember to change its generated clock in the SDC file. Examples of this type of change include modifying an existing output clock, adding a new PLL output, or making a change to the PLL's hierarchy. Too many designers forget to modify the SDC file and spend time debugging something that `derive_pll_clocks` would have updated automatically.

Related Links

- Creating Base Clocks on page 53
  Base clocks are the primary input clocks to the device.
- Deriving PLL Clocks on page 60
  Use the `derive_pll_clocks` command to direct the TimeQuest analyzer to automatically search the timing netlist for all unconstrained PLL output clocks. The `derive_pll_clocks` command detects your current PLL settings and automatically creates generated clocks on the outputs of every PLL by calling the `create_generated_clock` command.

3.3.1.3 derive_clock_uncertainty

Add the following command to your SDC file:

```
derive_clock_uncertainty
```

This command calculates clock-to-clock uncertainties within the FPGA, due to characteristics like PLL jitter, clock tree jitter, etc. This should be in all SDC files and the TimeQuest analyzer generates a warning if this command is not found in your SDC files.

Related Links

Accounting for Clock Effect Characteristics on page 63
The clocks you create with the TimeQuest analyzer are ideal clocks that do not account for any board effects. You can account for clock effect characteristics with clock latency and clock uncertainty.

3.3.1.4 SDC Constraint Creation Summary

This example shows the SDC file for a sample design with two clocks.

```
create_clock -period 20.00 -name adc_clk [get_ports adc_clk]
create_clock -period 8.00 -name sys_clk [get_ports sys_clk]

derive_pll_clocks

derive_clock_uncertainty
```

3.3.1.5 set_clock_groups

With the constraintsdiscusssed previously, most, if not all, of the clocks in the design are now constrained. In the TimeQuest analyzer, all clocks are related by default, and you must indicate which clocks are not related. For example, if there are paths between an 8ns clock and 10ns clock, even if the clocks are completely asynchronous, the TimeQuest analyzer attempts to meet a 2ns setup relationship between these clocks unless you indicate that they are not related. The TimeQuest analyzer analyzes everything known, rather than assuming that all clocks are unrelated and requiring that you relate them. The SDC language has a powerful constraint for setting unrelated clocks called `set_clock_groups`. A template for the the typical use of the `set_clock_groups` command is:

```
set_clock_groups -asynchronous -group {<clock1>...<clockn>} ... \
    -group {<clocka>...<clockn>}
```

The `set_clock_groups` command does not actually group clocks. Since the TimeQuest analyzer assumes all clocks are related by default, all clocks are effectively in one big group. Instead, the `set_clock_groups` command cuts timing between clocks in different groups.

There is no limit to the number of times you can specify a group option with `-group {<group of clocks>}`. When entering constraints through the GUI with **Edit ➤ Insert Constraint**, the **Set Clock Groups** dialog box only permits two clock groups, but this is only a limitation of that dialog box. You can always manually add more into the SDC file.

Any clock not listed in the assignment is related to all clocks. If you forget a clock, the TimeQuest analyzer acts conservatively and analyzes that clock in context with all other domains to which it connects.

The `set_clock_groups` command requires either the `-asynchronous` or `-exclusive` option. The `-asynchronous` flag means the clocks are both toggling, but not in a way that can synchronously pass data. The `-exclusive` flag means the clocks do not toggle at the same time, and hence are mutually exclusive. An example of this might be a clock multiplexor that has two generated clock assignments on its output. Since only one can toggle at a time, these clocks are `-exclusive`. TimeQuest does not currently analyze crosstalk explicitly. Instead, the timing models use extra guard bands to account for any potential crosstalk-induced delays. TimeQuest treats the `-asynchronous` and `-exclusive` options the same.

A clock cannot be within multiple `-group` groupings in a single assignment, however, you can have multiple `set_clock_groups` assignments.

Another way to cut timing between clocks is to use `set_false_path`. To cut timing between `sys_clk` and `dsp_clk`, a user might enter:

```
set_false_path -from [get_clocks sys_clk] -to [get_clocks
dsp_clk]

set_false_path -from [get_clocks dsp_clk] -to [sys_clk]
```

This works fine when there are only a few clocks, but quickly grows to a huge number of assignments that are completely unreadable. In a simple design with three PLLs that have multiple outputs, the `set_clock_groups` command can clearly show which clocks are related in less than ten lines, while `set_false_path` may be over 50 lines and be very non-intuitive on what is being cut.

Related Links

- Creating Generated Clocks on page 57
  Define generated clocks on any nodes in your design which modify the properties of a clock signal, including phase, frequency, offset, and duty cycle.
- Relaxing Setup with set_multicyle_path on page 74
  A common type of multicycle exception occurs when the data transfer rate is slower than the clock cycle.
- Accounting for a Phase Shift on page 75
  In this example, the design contains a PLL that performs a phase-shift on a clock whose domain exchanges data with domains that do not experience the phase shift.

3.3.1.5.1 Tips for Writing a set_clock_groups Constraint

Since **derive_pll_clocks** creates many of the clock names, you may not know all of the clock names to use in the clock groups.

A quick way to make this constraint is to use the SDC file you have created so far, with the three basic constraints described in previous topics. Make sure you have added it to your project, then open the TimeQuest timing analyzer GUI.

In the **Task** panel of the **TimeQuest**analyzer, double-click **Report Clocks**. This reads your existing SDC and applies it to your design, then reports all the clocks. From that report, highlight all of the clock names in the first column, and copy the names.

You have just copied all the clock names in your design in the exact format the TimeQuest analyzer recognizes. Paste them into your SDC file to make a list of all clock names, one per line..

Format that list into the `set_clock_groups` command by cutting and pasting clock names into appropriate groups. Then enter the following empty template in your SDC file::

```
set_clock_groups -asynchronous -group { \
} \
 -group { \
} \
-group  { \
} \
-group { \
}
```

Cut and paste clocks into groups to define how they’re related, adding or removing groups as necessary. Format to make the code readable.

Note: This command can be difficult to read on a single line. Instead, you should make use of the Tcl line continuation character "\". By putting a space after your last character and then "\", the end-of-line character is escaped. (And be careful not to have any whitespace after the escape character, or else it will escape the whitespace, not the end-of-line character).

```
set_clock_groups -asynchronous \
    -group {adc_clk \
       the_adc_pll|altpll_component_autogenerated|pll|clk[0] \
       the_adc_pll|altpll_component_autogenerated|pll|clk[1] \
       the_adc_pll|altpll_component_autogenerated|pll|clk[2] \
    } \
    -group {sys_clk \
       the_system_pll|altpll_component_autogenerated|pll|clk[0] \
       the_system_pll|altpll_component_autogenerated|pll|clk[1] \
    } \
    -group {the_system_pll|altpll_component_autogenerated|pll|clk[2] \
    }
```

Note: The last group has a PLL output `system_pll|..|clk[2]` while the input clock and other PLL outputs are in different groups. If PLLs are used, and the input clock frequency is not related to the frequency of the PLL's outputs, they must be treated asynchronously. Usually most outputs of a PLL are related and hence in the same group, but this is not a requirement, and depends on the requirements of your design.

For designs with complex clocking, writing this constraint can be an iterative process. For example, a design with two DDR3 cores and high-speed transceivers could easily have thirty or more clocks. In those cases, you can just add the clocks you’ve created. Since clocks not in the command are still related to every clock, you are conservatively grouping what is known. If there are still failing paths in the design

between unrelated clock domains, you can start adding in the new clock domains as necessary. In this case, a large number of the clocks won't actually be in the `set_clock_groups` command, since they are either cut in the SDC file for the IP core (such as the SDC files generated by the DDR3 cores), or they only connect to clock domains to which they are related.

For many designs, that is all that's necessary to constrain the core. Some common core constraints that will not be covered in this quick start section that user's do are:

- Add multicycles between registers which can be analyzed at a slower rate than the default analysis, in other words, increasing the time when data can be read, or 'opening the window'. For example, a 10ns clock period will have a 10ns setup relationship. If the data changes at a slower rate, or perhaps the registers toggle at a slower rate due to a clock enable, then you should apply a multicycle that relaxes the setup relationship (opens the the window so that valid data can pass). This is a multiple of the clock period, making the setup relationship 20ns, 40ns, etc., while keeping the hold relationship at 0ns. These types of multicycles are generally applied to paths.
- The second common form of multicycle is when the user wants to advance the cycle in which data is read, or 'shift the window'. This generally occurs when your design performs a small phase-shift on a clock. For example, if your design has two 10ns clocks exiting a PLL, but the second clock has a 0.5ns phase-shift, the default setup relationship from the main clock to the phase-shifted clock is 0.5ns and the hold relationship is -9.5ns. It is almost impossible to meet a 0.5ns setup relationship, and most likely you intend the data to transfer in the next window. By adding a multicycle from the main clock to the phase-shifted clock, the setup relationship becomes 10.5ns and the hold relationship becomes 0.5ns. This multicycle is generally applied between clocks and is something the user should think about as soon as they do a small phase-shift on a clock. This type of multicycle is called shifting the window.
- Add a `create_generated_clock` to ripple clocks. When a register's output drives the `clk` port of another register, that is a ripple clock. Clocks do not propagate through registers, so all ripple clocks must have a `create_generated_clock` constraint applied to them for correct analysis. Unconstrained ripple clocks appear in the **Report Unconstrained Paths** report, so they are easily recognized. In general, ripple clocks should be avoided for many reasons, and if possible, a clock enable should be used instead.
- Add a `create_generated_clock` to clock mux outputs. Without this, all clocks propagate through the mux and are related. TimeQuest analyze paths downstream from the mux where one clock input feeds the source register and the other clock input feeds the destination, and vice-versa. Although it could be valid, this is usually not preferred behavior. By putting `create_generated_clock` constraints on the mux output, which relates them to the clocks coming into the mux, you can correctly group these clocks with other clocks.

3.3.2 Creating Clocks and Clock Constraints

Clocks specify timing requirements for synchronous transfers and guide the Fitter optimization algorithms to achieve the best possible placement for your design. You must define all clocks and any associated clock characteristics, such as uncertainty or latency. The TimeQuest analyzer supports SDC commands that accommodate various clocking schemes such as:

- Base clocks
- Virtual clocks
- Multifrequency clocks
- Generated clocks

3.3.2.1 Creating Base Clocks

Base clocks are the primary input clocks to the device. Unlike clocks that are generated in the device (such as an on-chip PLL), base clocks are generated by off-chip oscillators or forwarded from an external device. Define base clocks at the top of your SDC file, because generated clocks and other constraints often reference base clocks. The TimeQuest timing analyzer ignores any constraints that reference a clock that has not been defined.

Use the `create_clock` command to create a base clock. Use other constraints, such as those described in *Accounting for Clock Effect Characteristics*, to specify clock characteristics such as uncertainty and latency.

The following examples show the most common uses of the `create_clock` constraint:

create_clock Command

To specify a 100 MHz requirement on a `clk_sys` input clock port you would enter the following in your SDC file:

```
create_clock -period 10 -name clk_sys [get_ports clk_sys]
```

100 MHz Shifted by 90 Degrees Clock Creation

This example creates a 10 ns clock with a 50% duty cycle that is phase shifted by 90 degrees applied to port `clk_sys`. This type of clock definition is most commonly used when the FPGA receives source synchronous, double-rate data that is center-aligned with respect to the clock.

```
create_clock -period 10 -waveform { 2.5 7.5 } [get_ports clk_sys]
```

Two Oscillators Driving the Same Clock Port

You can apply multiple clocks to the same target with the `-add` option. For example, to specify that the same clock input can be driven at two different frequencies, enter the following commands in your SDC file:

```
create_clock -period 10 -name clk_100 [get_ports clk_sys]
create_clock -period 5 -name clk_200 [get_ports clk_sys] -add
```

Although it is not common to have more than two base clocks defined on a port, you can define as many as are appropriate for your design, making sure you specify `-add` for all clocks after the first.

Creating Multifrequency Clocks

You must create a multifrequency clock if your design has more than one clock source feeding a single clock node in your design. The additional clock may act as a low-power clock, with a lower frequency than the primary clock. If your design uses multifrequency clocks, use the `set_clock_groups` command to define clocks that are exclusive.

To create multifrequency clocks, use the `create_clock` command with the `-add` option to create multiple clocks on a clock node. You can create a 10 ns clock applied to clock port `clk`, and then add an additional 15 ns clock to the same clock port. The TimeQuest analyzer uses both clocks when it performs timing analysis.

```
create_clock –period 10 –name clock_primary –waveform { 0 5 } \
    [get_ports clk]
create_clock –period 15 –name clock_secondary –waveform { 0 7.5 } \
    [get_ports clk] -add
```

Related Links

- Accounting for Clock Effect Characteristics on page 63
  The clocks you create with the TimeQuest analyzer are ideal clocks that do not account for any board effects. You can account for clock effect characteristics with clock latency and clock uncertainty.
- create_clock
- get_ports
  For more information about these commands, refer to Quartus Prime Help.

3.3.2.1.1 Automatically Detecting Clocks and Creating Default Clock Constraints

To automatically create base clocks in your design, use the `derive_clocks` command. The `derive_clocks` command is equivalent to using the `create_clock` command for each register or port feeding the clock pin of a register. The `derive_clocks` command creates clock constraints on ports or registers to ensure every register in your design has a clock constraints, and it applies one period to all base clocks in your design.

You can have the TimeQuest analyzer create a base clock with a 100 Mhz requirement for unconstrained base clock nodes.

```
derive_clocks -period 10
```

Warning: Do not use the `derive_clocks` command for final timing sign-off; instead, you should create clocks for all clock sources with the `create_clock` and `create_generated_clock` commands. If your design has more than a single clock, the `derive_clocks` command constrains all the clocks to the same specified frequency. To achieve a thorough and realistic analysis of your design's timing requirements, you should make individual clock constraints for all clocks in your design.

If you want to have some base clocks created automatically, you can use the `-create_base_clocks` option to `derive_pll_clocks`. With this option, the `derive_pll_clocks` command automatically creates base clocks for each PLL, based on the input frequency information specified when the PLL was instantiated. The

base clocks are named matching the port names. This feature works for simple port-to-PLL connections. Base clocks are not automatically generated for complex PLL connectivity, such as cascaded PLLs. You can also use the command `derive_pll_clocks -create_base_clocks` to create the input clocks for all PLL inputs automatically.

Related Links

derive_clocks
For more information about this command, refer to Quartus Prime Help.

3.3.2.2 Creating Virtual Clocks

A virtual clock is a clock that does not have a real source in the design or that does not interact directly with the design.

To create virtual clocks, use the `create_clock` command with no value specified for the <*targets*> option.

This example defines a 100Mhz virtual clock because no target is specified.

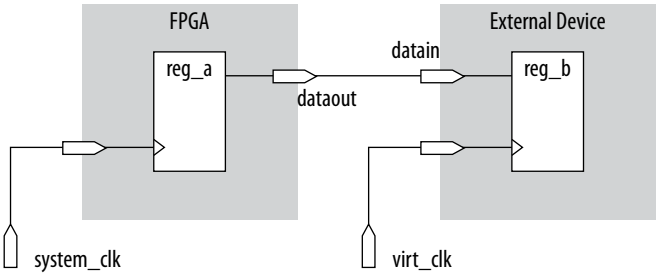
```
create_clock -period 10 -name my_virt_clk
```

I/O Constraints with Virtual Clocks

Virtual clocks are most commonly used in I/O constraints; they represent the clock at the external device connected to the FPGA.

For the output circuit shown in the following figure, you should use a base clock to constrain the circuit in the FPGA, and a virtual clock to represent the clock driving the external device. Examples of the base clock, virtual clock, and output delay constraints for such a circuit are shown below.

Figure 37. Virtual Clock Board Topology



You can create a 10 ns virtual clock named `virt_clk` with a 50% duty cycle where the first rising edge occurs at 0 ns by adding the following code to your SDC file. The virtual clock is then used as the clock source for an output delay constraint.

Example 2. Virtual Clock

```
#create base clock for the design
create_clock -period 5 [get_ports system_clk]
#create the virtual clock for the external register
create_clock -period 10 -name virt_clk
```

```
#set the output delay referencing the virtual clock
set_output_delay -clock virt_clk -max 1.5 [get_ports dataout]
set_output_delay -clock virt_clk -min 0.0 [get_ports dataout]
```

Related Links

- set_input_delay
- set_output_delay

  For more information about these commands, refer to Quartus Prime Help.

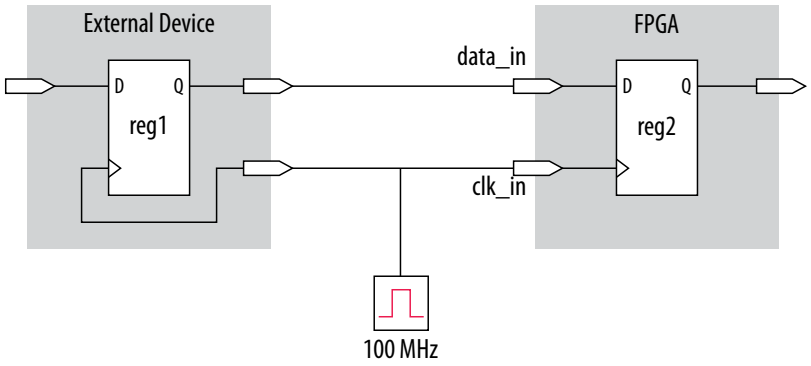
3.3.2.2.1 Example of Specifying an I/O Interface Clock

To specify I/O interface uncertainty, you must create a virtual clock and constrain the input and output ports with the `set_input_delay` and `set_output_delay` commands that reference the virtual clock.

When the `set_input_delay` or `set_output_delay` commands reference a clock port or PLL output, the virtual clock allows the `derive_clock_uncertainty` command to apply separate clock uncertainties for internal clock transfers and I/O interface clock transfers

Create the virtual clock with the same properties as the original clock that is driving the I/O port.

Figure 38. I/O Interface Clock Specifications



Example 3. SDC Commands to Constrain the I/O Interface

```
# Create the base clock for the clock port
create_clock -period 10 -name clk_in [get_ports clk_in]
# Create a virtual clock with the same properties of the base clock
# driving the source register
create_clock -period 10 -name virt_clk_in
# Create the input delay referencing the virtual clock and not the base
# clock
# DO NOT use set_input_delay -clock clk_in <delay value>
# [get_ports data_in]
set_input_delay -clock virt_clk_in <delay value> [get_ports data_in]
```

3.3.2.3 I/O Interface Uncertainty

Virtual clocks are recommended for I/O constraints because they most accurately represent the clocking topology of the design. An additional benefit is that you can specify different uncertainty values on clocks that interface with external I/O ports and clocks that feed register-to-register paths inside the FPGA.

Related Links

Clock Uncertainty on page 64
For more information about clock uncertainty and clock transfers.

3.3.2.4 Creating Generated Clocks

Define generated clocks on any nodes in your design which modify the properties of a clock signal, including phase, frequency, offset, and duty cycle. Generated clocks are most commonly used on the outputs of PLLs, on register clock dividers, clock muxes, and clocks forwarded to other devices from an FPGA output port, such as source synchronous and memory interfaces. In the SDC file, create generated clocks after the base clocks have been defined. Generated clocks automatically account for all clock delays and clock latency to the generated clock target.

Use the `create_generated_clock` command to constrain generated clocks in your design.

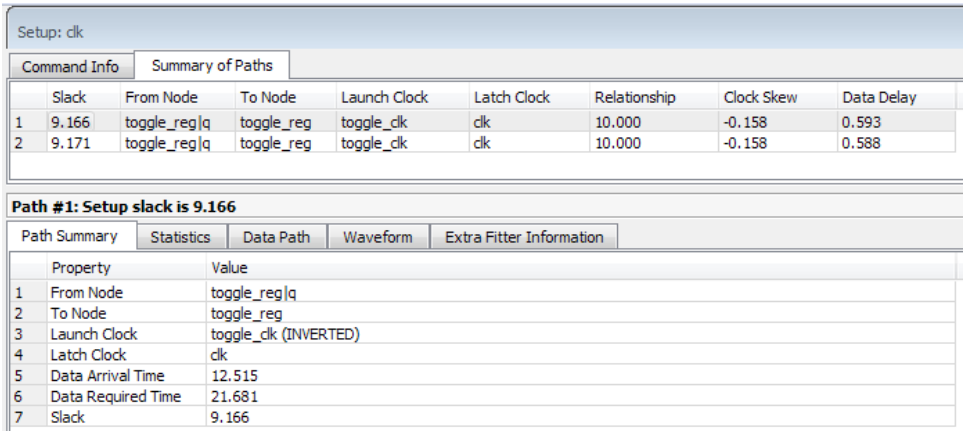
The `-source` option specifies the name of a node in the clock path which is used as reference for your generated clock. The source of the generated clock must be a node in your design netlist and not the name of a previously defined clock. You can use any node name on the clock path between the input clock pin of the target of the generated clock and the target node of its reference clock as the source node. A good practice is to specify the input clock pin of the target node as the source of your new generated clock. That way, the source of the generated clock is decoupled from the naming and hierarchy of its clock source. If you change its clock source, you don't have to edit the generated clock constraint.

If you have multiple base clocks feeding a node that is the source for a generated clock, you must define multiple generated clocks. Each generated clock is associated to one base clock using the `-master_clock` option in each generated clock statement. In some cases, generated clocks are generated with combinational logic. Depending on how your clock-modifying logic is synthesized, the name can change from compile to compile. If the name changes after you write the generated clock constraint, the generated clock is ignored because its target name no longer exists in the design. To avoid this problem, use a synthesis attribute or synthesis assignment to keep the final combinational node of the clock-modifying logic. Then use the kept name in your generated clock constraint. For details on keeping combinational nodes or wires, refer to the *Implement as Output of Logic Cell logic option* topic in Quartus Prime Help.

When a generated clock is created on a node that ultimately feeds the data input of a register, this creates a special case referred to as “clock-as-data”. Instances of clock-as-data are treated differently by TimeQuest. For example, when clock-as-data is used with DDR, both the rise and the fall of this clock need to be considered since it is a clock, and TimeQuest reports both rise and fall. With clock-as-data, the **From Node** is treated as the target of the generated clock, and the **Launch Clock** is treated as the generated clock. In the figure below, the first path is from **toggle_clk (INVERTED)** to **clk**, whereas the second path is from **toggle_clk** to **clk**. The slack in both cases is slightly different due to the difference in rise and fall times along the path; the ~5 ps

difference can be seen in the **Data Delay** column. Only the path with the lowest slack value need be considered. This would also be true if this were not a clock-as-data case, but normally TimeQuest only reports the worst-case path between the two (rise and fall). In this example, if the generated clock were not defined on the register output, then only one path would be reported and it would be the one with the lowest slack value. If your design targets an Arria 10 device, the enhanced timing algorithms remove all common clock pessimism on paths treated as clock-as-data.

Figure 39. Example of clock-as-data



The TimeQuest analyzer provides the `derive_pll_clocks` command to automatically generate clocks for all PLL clock outputs. The properties of the generated clocks on the PLL outputs match the properties defined for the PLL.

Related Links

- Deriving PLL Clocks on page 60
  For more information about deriving PLL clock outputs.
- Implement as Output of Logic Cell logic option
  For more information on keeping combinational nodes or wires, refer to Quartus Prime Help.
- create_generate_clock
- derive_pll_clocks
- create_generated_clocks
  For information about these commands, refer to Quartus Prime Help.

3.3.2.4.1 Clock Divider Example

A common form of generated clock is a divide-by-two register clock divider. The following constraint creates a half-rate clock on the divide-by-two register.

Figure 40. Clock Divider

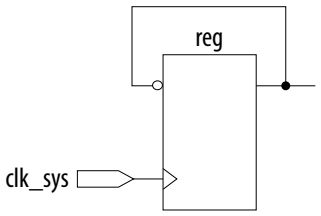
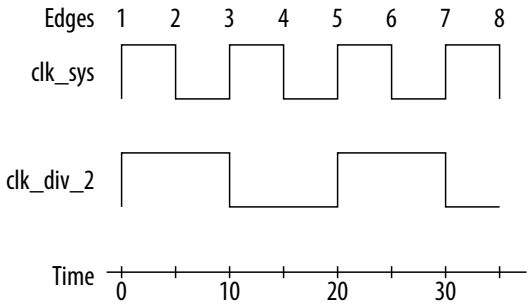


Figure 41. Clock Divider Waveform




```
create_clock -period 10 -name clk_sys [get_ports clk_sys]
create_generated_clock -name clk_div_2 -divide_by 2 -source \
    [get_ports clk_sys] [get_pins reg|q]
```

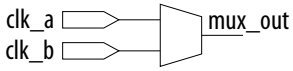
Or in order to have the clock source be the clock pin of the register you can use:

```
create_clock -period 10 -name clk_sys [get_ports clk_sys]
create_generated_clock -name clk_div_2 -divide_by 2 -source \
    [get_pins reg|clk] [get_pins reg|q]
```

3.3.2.4.2 Clock Multiplexor Example

Another common form of generated clock is on the output of a clock mux. One generated clock on the output is requred for each input clock. The SDC example also includes the `set_clock_groups` command to indicate that the two generated clocks can never be active simultaneously in the design, so the TimeQuest analyzer does not analyze cross-domain paths between the generated clocks on the output of the clock mux.

Figure 42. Clock Mux




```
create_clock -name clock_a -period 10 [get_ports clk_a]
create_clock -name clock_b -period 10 [get_ports clk_b]
create_generated_clock -name clock_a_mux -source [get_ports clk_a] \
    [get_pins clk_mux|mux_out]
create_generated_clock -name clock_b_mux -source [get_ports clk_b] \
    [get_pins clk_mux|mux_out] -add
set_clock_groups -exclusive -group clock_a_mux -group clock_b_mux
```

3.3.2.5 Deriving PLL Clocks

Use the `derive_pll_clocks` command to direct the TimeQuest analyzer to automatically search the timing netlist for all unconstrained PLL output clocks. The `derive_pll_clocks` command detects your current PLL settings and automatically creates generated clocks on the outputs of every PLL by calling the `create_generated_clock` command.

Create Base Clock for PLL input Clock Ports

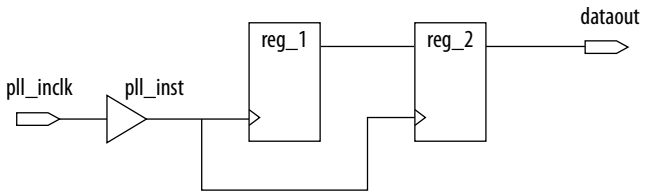
```
create_clock -period 10.0 -name fpga_sys_clk [get_ports fpga_sys_clk] \
    derive_pll_clocks
```

If your design contains transceivers, LVDS transmitters, or LVDS receivers, you must use the `derive_pll_clocks` command. The command automatically constrains this logic in your design and creates timing exceptions for those blocks.

Include the `derive_pll_clocks` command in your SDC file after any `create_clock` command Each time the TimeQuest analyzer reads your SDC file, the appropriate generate clock is created for each PLL output clock pin. If a clock exists on a PLL output before running `derive_pll_clocks`, the pre-existing clock has precedence, and an auto-generated clock is not created for that PLL output.

A simple PLL design with a register-to-register path.

Figure 43. Simple PLL Design



The TimeQuest analyzer generates messages when you use the `derive_pll_clocks` command to automatically constrain the PLL for a design similar to the previous image.

Example 4. derive_pll_clocks Command Messages

```
Info:
Info: Deriving PLL Clocks:
Info: create_generated_clock -source pll_inst|altpll_component|pll|inclk[0] -
divide_by 2 -name
pll_inst|altpll_component|pll|clk[0] pll_inst|altpll_component|pll|clk[0]
Info:
```

The input clock pin of the PLL is the node `pll_inst|altpll_component|pll|inclk[0]` which is used for the `-source` option. The name of the output clock of the PLL is the PLL output clock node, `pll_inst|altpll_component|pll|clk[0]`.

If the PLL is in clock switchover mode, multiple clocks are created for the output clock of the PLL; one for the primary input clock (for example, `inclk[0]`), and one for the secondary input clock (for example, `inclk[1]`). You should create exclusive clock groups for the primary and secondary output clocks since they are not active simultaneously.

Related Links

- Creating Clock Groups on page 61
  For more information about creating exclusive clock groups.
- derive_pll_clocks
- Derive PLL Clocks
  For more information about the derive_pll_clocks command.

3.3.2.6 Creating Clock Groups

The TimeQuest analyzer assumes all clocks are related unless constrained otherwise.

To specify clocks in your design that are exclusive or asynchronous, use the `set_clock_groups` command. The `set_clock_groups` command cuts timing between clocks in different groups, and performs the same analysis regardless of whether you specify `-exclusive` or `-asynchronous`. A group is defined with the `-group` option. The TimeQuest analyzer excludes the timing paths between clocks for each of the separate groups.

The following tables show examples of various group options for the `set_clock_groups` command.

Table 10. set_clock_groups -group A

Dest\Source	A	B	C	D
A	Analyzed	Cut	Cut	Cut
B	Cut	Analyzed	Analyzed	Analyzed
C	Cut	Analyzed	Analyzed	Analyzed
D	Cut	Analyzed	Analyzed	Analyzed

Table 11. set_clock_groups -group {A B}

Dest\Source	A	B	C	D
A	Analyzed	Analyzed	Cut	Cut
B	Analyzed	Analyzed	Cut	Cut
C	Cut	Cut	Analyzed	Analyzed
D	Cut	Cut	Analyzed	Analyzed

Table 12. set_clock_groups -group A -group B

Dest\Source	A	B	C	D
A	Analyzed	Cut	Cut	Cut

continued...

B	Cut	Analyzed	Cut	Cut
C	Cut	Cut	Analyzed	Analyzed
D	Cut	Cut	Analyzed	Analyzed

Table 13. set_clock_groups -group {A C} -group {B D}

Dest\Source	A	B	C	D
A	Analyzed	Cut	Analyzed	Cut
B	Cut	Analyzed	Cut	Analyzed
C	Analyzed	Cut	Analyzed	Cut
D	Cut	Analyzed	Cut	Analyzed

Table 14. set_clock_groups -group {A C D}

Dest\Source	A	B	C	D
A	Analyzed	Cut	Analyzed	Analyzed
B	Cut	Analyzed	Cut	Cut
C	Analyzed	Cut	Analyzed	Analyzed
D	Analyzed	Cut	Analyzed	Analyzed

Related Links

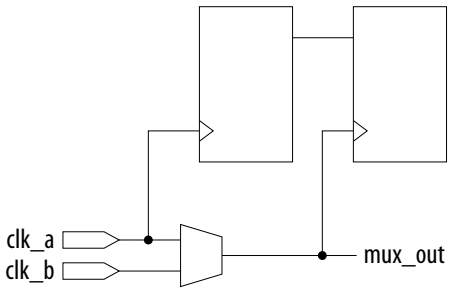
set_clock_groups
For more information about this command, refer to Quartus Prime Help.

3.3.2.6.1 Exclusive Clock Groups

Use the `-exclusive` option to declare that two clocks are mutually exclusive. You may want to declare clocks as mutually exclusive when multiple clocks are created on the same node. This case occurs for multiplexed clocks.

For example, an input port may be clocked by either a 25-MHz or a 50-MHz clock. To constrain this port, create two clocks on the port, and then create clock groups to declare that they do not coexist in the design at the same time. Declaring the clocks as mutually exclusive eliminates clock transfers that are derived between the 25-MHz clock and the 50-MHz clock.

Figure 44. Clock Mux with Synchronous Path Across the Mux




```
create_clock -period 40 -name clk_a [get_ports {port_a}]
create_clock -add -period 20 -name clk_b [get_ports {port_a}]
set_clock_groups -exclusive -group {clk_a} -group {clk_b}
```

3.3.2.6.2 Asynchronous Clock Groups

Use the `-asynchronous` option to create asynchronous clock groups. Asynchronous clock groups are commonly used to break the timing relationship where data is transfered through a FIFO between clocks running at different rates.

Related Links

set_clock_groups
For more information about this command, refer to Quartus Prime Help.

3.3.2.7 Accounting for Clock Effect Characteristics

The clocks you create with the TimeQuest analyzer are ideal clocks that do not account for any board effects. You can account for clock effect characteristics with clock latency and clock uncertainty.

3.3.2.7.1 Clock Latency

There are two forms of clock latency, clock source latency and clock network latency. Source latency is the propagation delay from the origin of the clock to the clock definition point (for example, a clock port). Network latency is the propagation delay from a clock definition point to a register's clock pin. The total latency at a register's clock pin is the sum of the source and network latencies in the clock path.

To specify source latency to any clock ports in your design, use the `set_clock_latency` command.

Note: The TimeQuest analyzer automatically computes network latencies; therefore, you only can characterize source latency with the `set_clock_latency` command. You must use the `-source` option.

Related Links

set_clock_latency
For more information about this command, refer to Quartus Prime Help.

3.3.2.7.2 Clock Uncertainty

When clocks are created, they are ideal and have perfect edges. It is important to add uncertainty to those perfect edges, to mimic clock-level effects like jitter. You should include the `derive_clock_uncertainty` command in your SDC file so that appropriate setup and hold uncertainties are automatically calculated and applied to all clock transfers in your design. If you don't include the command, the TimeQuest analyzer performs it anyway; it is a critical part of constraining your design correctly.

The TimeQuest analyzer subtracts setup uncertainty from the data required time for each applicable path and adds the hold uncertainty to the data required time for each applicable path. This slightly reduces the setup and hold slack on each path.

The TimeQuest analyzer accounts for uncertainty clock effects for three types of clock-to-clock transfers; intraclock transfers, interclock transfers, and I/O interface clock transfers.

- Intraclock transfers occur when the register-to-register transfer takes place in the device and the source and destination clocks come from the same PLL output pin or clock port.
- Interclock transfers occur when a register-to-register transfer takes place in the core of the device and the source and destination clocks come from a different PLL output pin or clock port.
- I/O interface clock transfers occur when data transfers from an I/O port to the core of the device or from the core of the device to the I/O port.

To manually specify clock uncertainty, use the `set_clock_uncertainty` command. You can specify the uncertainty separately for setup and hold. You can also specify separate values for rising and falling clock transitions, although this is not commonly used. You can override the value that was automatically applied by the `derive_clock_uncertainty` command, or you can add to it.

The `derive_clock_uncertainty` command accounts for PLL clock jitter if the clock jitter on the input to a PLL is within the input jitter specification for PLL's in the specified device. If the input clock jitter for the PLL exceeds the specification, you should add additional uncertainty to your PLL output clocks to account for excess jitter with the `set_clock_uncertainty -add` command. Refer to the device handbook for your device for jitter specifications.

Another example is to use `set_clock_uncertainty -add` to add uncertainty to account for peak-to-peak jitter from a board when the jitter exceeds the jitter specification for that device. In this case you would add uncertainty to both setup and hold equal to 1/2 the jitter value:

```
set_clock_uncertainty –setup –to <clock name>  \
    -setup –add <p2p jitter/2>
```

```
set_clock_uncertainty –hold –enable_same_physical_edge –to <clock name> \
    –add <p2p jitter/2>
```

There is a complex set of precedence rules for how the TimeQuest analyzer applies values from `derive_clock_uncertainty` and `set_clock_uncertainty`, which depend on the order the commands appear in your SDC files, and various options

used with the commands. The Help topics referred to below contain complete descriptions of these rules. These precedence rules are much simpler to understand and implement if you follow these recommendations:

- If you want to assign your own clock uncertainty values to any clock transfers, the best practice is to put your `set_clock_uncertainty` exceptions after the `derive_clock_uncertainty` command in your SDC file.
- When you use the `-add` option for `set_clock_uncertainty`, the value you specify is added to the value from `derive_clock_uncertainty`. If you don't specify `-add`, the value you specify replaces the value from `derive_clock_uncertainty`.

Related Links

- set_clock_uncertainty
- derive_clock_uncertainty
- remove_clock_uncertainty

  For more information about these commands, refer to Quartus Prime Help.

3.3.3 Creating I/O Requirements

The TimeQuest analyzer reviews setup and hold relationships for designs in which an external source interacts with a register internal to the design. The TimeQuest analyzer supports input and output external delay modeling with the `set_input_delay` and `set_output_delay` commands. You can specify the clock and minimum and maximum arrival times relative to the clock.

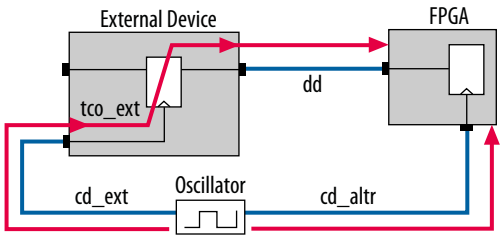
You must specify timing requirements, including internal and external timing requirements, before you fully analyze a design. With external timing requirements specified, the TimeQuest analyzer verifies the I/O interface, or periphery of the device, against any system specification.

3.3.3.1 Input Constraints

Input constraints allow you to specify all the external delays feeding into the device. Specify input requirements for all input ports in your design.

You can use the `set_input_delay` command to specify external input delay requirements. Use the `-clock` option to reference a virtual clock. Using a virtual clock allows the TimeQuest analyzer to correctly derive clock uncertainties for interclock and intraclock transfers. The virtual clock defines the launching clock for the input port. The TimeQuest analyzer automatically determines the latching clock inside the device that captures the input data, because all clocks in the device are defined.

Figure 45. Input Delay



The calculation the TimeQuest analyzer performs to determine the typical input delay.

Figure 46. Input Delay Calculation

$$\text{input delay}_{MAX} = (\text{cd\_ext}_{MAX} - \text{cd\_altr}_{MIN}) + \text{tco\_ext}_{MAX} + \text{dd}_{MAX}$$
$$\text{input delay}_{MIN} = (\text{cd\_ext}_{MIN} - \text{cd\_altr}_{MAX}) + \text{tco\_ext}_{MIN} + \text{dd}_{MIN}$$

If your design features partition boundary ports, you can use the `-blackbox` option with `set_input_delay`to assign input delays. The `-blackbox` option creates a new keeper timing node with the same name as the boudary port. This new node permits the propagation of timing paths through the original boundary port and acts as a `set_input_delay` constraint. The new keeper timing nodes are displayed when you use the `get_keepers` SDC command.

You can remove these blackbox constraints with `remove_input_constraint -blackbox`.

Related Links

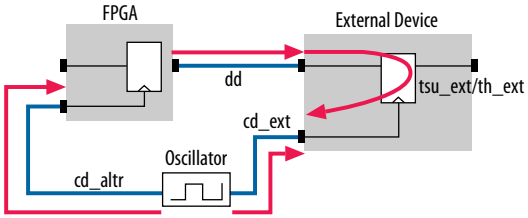
SDC (Clock and Exception) Assignments on Blackbox Ports on page 95
In the Quartus Prime Pro Edition software you can assign clock definitions and SDC exceptions on partition ports. Your design should follow the Hierarchical Design flow as a prerequisite.

3.3.3.2 Output Constraints

Output constraints allow you to specify all external delays from the device for all output ports in your design.

You can use the `set_output_delay` command to specify external output delay requirements. Use the `-clock` option to reference a virtual clock. The virtual clock defines the latching clock for the output port. The TimeQuest analyzer automatically determines the launching clock inside the device that launches the output data, because all clocks in the device are defined. The following figure is an example of an output delay referencing a virtual clock.

Figure 47. Output Delay



The calculation the TimeQuest analyzer performs to determine the typical out delay.

Figure 48. Output Delay Calculation

$$\text{output delay}_{MAX} = \text{dd}_{MAX} + \text{tsu\_ext} + (\text{cd\_altr}_{MAX} - \text{cd\_ext}_{MIN})$$
$$\text{output delay}_{MIN} = (\text{dd}_{MIN} - \text{th\_ext} + (\text{cd\_altr}_{MIN} - \text{cd\_ext}_{MAX}))$$

If your design features partition boundary ports, you can use the `-blackbox` option with `set_ouput_delay`to assign output delays. The `-blackbox` option creates a new keeper timing node with the same name as the boudary port. This new node permits the propagation of timing paths through the original boundary port and acts as a `set_output_delay` constraint. The new keeper timing nodes are displayed when you use the `get_keepers` SDC command.

You can remove these blackbox constraints with `remove_output_constraint -blackbox`.

Related Links

- SDC (Clock and Exception) Assignments on Blackbox Ports on page 95
  In the Quartus Prime Pro Edition software you can assign clock definitions and SDC exceptions on partition ports. Your design should follow the Hierarchical Design flow as a prerequisite.
- set_intput_delay
- set_output_delay
  For more information about these commands, refer to Quartus Prime Help.

3.3.4 Creating Delay and Skew Constraints

The TimeQuest analyzer supports the Synopsys Design Constraint format for constraining timing for the ports in your design. These constraints allow the TimeQuest analyzer to perform a system static timing analysis that includes not only the device internal timing, but also any external device timing and board timing parameters.

3.3.4.1 Advanced I/O Timing and Board Trace Model Delay

The TimeQuest analyzer can use advanced I/O timing and board trace model assignments to model I/O buffer delays in your design.

If you change any advanced I/O timing settings or board trace model assignments, recompile your design before you analyze timing, or use the `-force_dat` option to force delay annotation when you create a timing netlist.

Example 5. Forcing Delay Annotation

```
create_timing_netlist -force_dat
```

Related Links

- Using Advanced I/O Timing
- I/O Management
  For more information about advanced I/O timing.

3.3.4.2 Maximum Skew

To specify the maximum path-based skew requirements for registers and ports in the design and report the results of maximum skew analysis, use the `set_max_skew` command in conjunction with the `report_max_skew` command.

Use the `set_max_skew` constraint to perform maximum allowable skew analysis between sets of registers or ports. In order to constrain skew across multiple paths, all such paths must be defined within a single `set_max_skew` constraint. `set_max_skew` timing constraint is not affected by **set_max_delay**, `set_min_delay`, and `set_multicycle_path` but it does obey `set_false_path` and `set_clock_groups`. If your design targets an Arria 10 device, skew constraints are not affected by `set_clock_groups`.

Table 15. set_max_skew Options

Arguments	Description
`-h | -help`	Short help
`-long_help`	Long help with examples and possible return values
`-exclude <Tcl list>`	A Tcl list of parameters to exclude during skew analysis. This list can include one or more of the following: `utsu`, `uth`, `utco`, `from_clock`, `to_clock`, `clock_uncertainty`, `ccpp`, `input_delay`, `output_delay`, `odv`. *Note:* Not supported for Arria 10 devices.
`-fall_from_clock <names>`	Valid source clocks (string patterns are matched using Tcl string matching)
`-fall_to_clock <names>`	Valid destination clocks (string patterns are matched using Tcl string matching)
`-from <names>`[1]	Valid sources (string patterns are matched using Tcl string matching
`-from_clock <names>`	Valid source clocks (string patterns are matched using Tcl string matching)
`-get_skew_value_from_clock_period <src_clock_period|dst_clock_period|min_clock_period>`	Option to interpret skew constraint as a multiple of the clock period

continued...

1 Legal values for the -from and -to options are collections of clocks, registers, ports, pins, cells or partitions in a design.

Arguments	Description
-include <*Tcl list*>	Tcl list of parameters to include during skew analysis. This list can include one or more of the following: utsu, uth, utco, from_clock, to_clock, clock_uncertainty, ccpp, input_delay, output_delay, odv. *Note:* Not supported for Arria 10 devices.
-rise_from_clock <*names*>	Valid source clocks (string patterns are matched using Tcl string matching)
-rise_to_clock <*names*>	Valid destination clocks (string patterns are matched using Tcl string matching)
-skew_value_multiplier <*multiplier*>	Value by which the clock period should be multiplied to compute skew requirement.
-to <names>[1]	Valid destinations (string patterns are matched using Tcl string matching)
-to_clock <*names*>	Valid destination clocks (string patterns are matched using Tcl string matching)
<*skew*>	Required skew

Applying maximum skew constraints between clocks applies the constraint from all register or ports driven by the clock specified with the -from option to all registers or ports driven by the clock specified with the -to option.

Use the -include and -exclude options to include or exclude one or more of the following: register micro parameters (utsu, uth, utco), clock arrival times (from_clock, to_clock), clock uncertainty (clock_uncertainty), common clock path pessimism removal (ccpp), input and output delays (input_delay, output_delay) and on-die variation (odv). Max skew analysis can include data arrival times, clock arrival times, register micro parameters, clock uncertainty, on-die variation and ccpp removal. Among these, only ccpp removal is disabled during the Fitter by default. When -include is used, those in the inclusion list are added to the default analysis. Similarly, when -exclude is used, those in the exclusion list are excluded from the default analysis. When both the -include and -exclude options specify the same parameter, that parameter is excluded.

Note: If your design targets an Arria 10 device, -exclude and -include are not supported.

Use -get_skew_value_from_clock_period to set the skew as a multiple of the launching or latching clock period, or whichever of the two has a smaller period. If this option is used, then -skew_value_multiplier must also be set, and the positional skew option may not be set. If the set of skew paths is clocked by more than one clock, TimeQuest uses the one with smallest period to compute the skew constraint.

When this constraint is used, results of max skew analysis are displayed in the Report Max Skew (report_max_skew) report from the TimeQuest Timing Analyzer. Since skew is defined between two or more paths, no results are displayed if the -from/-from_clock and -to/-to_clock filters satisfy less than two paths.

Related Links

- set_max_skew
- report_max_skew

For more information about these commands, refer to Quartus Prime Help.

3.3.4.3 Net Delay

Use the `set_net_delay` command to set the net delays and perform minimum or maximum timing analysis across nets.

The `-from` and `-to` options can be string patterns or pin, port, register, or net collections. When pin or net collection is used, the collection should include output pins or nets.

Table 16. set_net_delay Options

Arguments	Description
`-h | -help`	Short help
`-long_help`	Long help with examples and possible return values
`-from <names>`	Valid source pins, ports, registers or nets(string patterns are matched using Tcl string matching)
`-get_value_from_clock_period <src_clock_period|dst_clock_period|min_clock_period|max_clock_period>`	Option to interpret net delay constraint as a multiple of the clock period.
`-max`	Specifies maximum delay
`-min`	Specifies minimum delay
`-to <names>`[2]	Valid destination pins, ports, registers or nets (string patterns are matched using Tcl string matching)
`-value_multiplier <multiplier>`	Value by which the clock period should be multiplied to compute net delay requirement.
`<delay>`	Required delay

When you use the -min option, slack is calculated by looking at the minimum delay on the edge. If you use -max option, slack is calculated with the maximum edge delay.

Use `-get_skew_value_from_clock_period` to set the net delay requirement as a multiple of the launching or latching clock period, or whichever of the two has a smaller or larger period. If this option is used, then `-value_multiplier` must also be set, and the positional delay option may not be set. If the set of nets is clocked by more than one clock, TimeQuest uses the net with smallest period to compute the constraint for a `-max` constraint, and the largest period for a `-min` constraint. If there are no clocks clocking the endpoints of the net (e.g. if the endpoints of the nets are not registers or constraint ports), then the net delay constraint will be ignored.

Related Links

- set_net_delay
- report_net_delay
  For more information about these commands, refer to Quartus Prime Help.

2 If the -to option is unused or if the `-to` filter is a wildcard ("*") character, all the output pins and registers on timing netlist became valid destination points.

3.3.4.4 Using create_timing_netlist

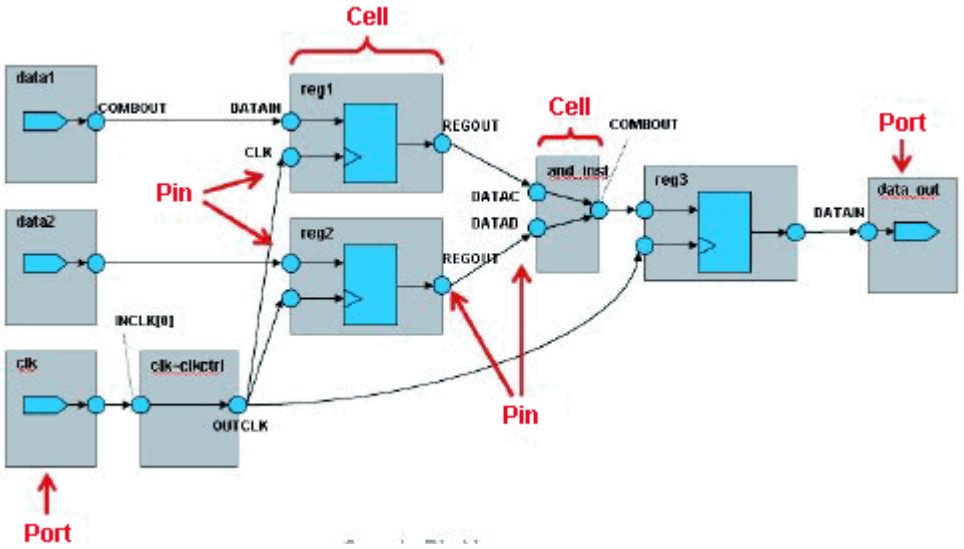
You can onfigure or load the timing netlist that the TimeQuest analyzer uses to calculate path delay data.

Your design should have a timing netlist before running the TimeQuest analyzer. You can use the **Create Timing Netlist** dialog box or the **Create Timing Netlist** command in the **Tasks** pane. The command also generates Advanced I/O Timing reports if you turned on **Enable Advanced I/O Timing** in the **TimeQuest Timing Analyzer** page of the **Settings** dialog box.

Note: The timing netlist created is based on the initial configuration of the design. Any configuration changes done by the design after the device enters user mode, for example, dynamic transceiver reconfiguration, are not reflected in the timing netlist. This applies to all device families except transceivers on Arria 10 devices with the Multiple Reconfiguration Profiles feature.

The following diagram shows how the TimeQuest analyzer interprets and classifies timing netlist data for a sample design.

Figure 49. How TimeQuest Interprets the Timing Netlist



3.3.5 Creating Timing Exceptions

Timing exceptions in the TimeQuest analyzer provide a way to modify the default timing analysis behavior to match the analysis required by your design. Specify timing exceptions after clocks and input and output delay constraints because timing exceptions can modify the default analysis.

3.3.5.1 Precedence

If a conflict of node names occurs between timing exceptions, the following order of precedence applies:

1. False path
2. Minimum delays and maximum delays
3. Multicycle path

The false path timing exception has the highest precedence. Within each category, assignments to individual nodes have precedence over assignments to clocks. Finally, the remaining precedence for additional conflicts is order-dependent, such that the assignments most recently created overwrite, or partially overwrite, earlier assignments.

3.3.5.2 False Paths

Specifying a false path in your design removes the path from timing analysis.

Use the `set_false_path` command to specify false paths in your design. You can specify either a point-to-point or clock-to-clock path as a false path. For example, a path you should specify as false path is a static configuration register that is written once during power-up initialization, but does not change state again. Although signals from static configuration registers often cross clock domains, you may not want to make false path exceptions to a clock-to-clock path, because some data may transfer across clock domains. However, you can selectively make false path exceptions from the static configuration register to all endpoints.

To make false path exceptions from all registers beginning with A to all registers beginning with B, use the following code in your SDC file.

```
set_false_path -from [get_pins A*] -to [get_pins B*]
```

The TimeQuest analyzer assumes all clocks are related unless you specify otherwise. Clock groups are a more efficient way to make false path exceptions between clocks, compared to writing multiple `set_false_path` exceptions between every clock transfer you want to eliminate.

Related Links

- Creating Clock Groups on page 61
  For more information about creating exclusive clock groups.
- set_false_path
  For more information about this command, refer to Quartus Prime Help.

3.3.5.3 Minimum and Maximum Delays

To specify an absolute minimum or maximum delay for a path, use the `set_min_delay` command or the `set_max_delay` commands, respectively. Specifying minimum and maximum delay directly overwrites existing setup and hold relationships with the minimum and maximum values.

Use the `set_max_delay` and `set_min_delay` commands to create constraints for asynchronous signals that do not have a specific clock relationship in your design, but require a minimum and maximum path delay. You can create minimum and maximum delay exceptions for port-to-port paths through the device without a register stage in the path. If you use minimum and maximum delay exceptions to constrain the path delay, specify both the minimum and maximum delay of the path; do not constrain only the minimum or maximum value.

If the source or destination node is clocked, the TimeQuest analyzer takes into account the clock paths, allowing more or less delay on the data path. If the source or destination node has an input or output delay, that delay is also included in the minimum or maximum delay check.

If you specify a minimum or maximum delay between timing nodes, the delay applies only to the path between the two nodes. If you specify a minimum or maximum delay for a clock, the delay applies to all paths where the source node or destination node is clocked by the clock.

You can create a minimum or maximum delay exception for an output port that does not have an output delay constraint. You cannot report timing for the paths associated with the output port; however, the TimeQuest analyzer reports any slack for the path in the setup summary and hold summary reports. Because there is no clock associated with the output port, no clock is reported for timing paths associated with the output port.

Note: To report timing with clock filters for output paths with minimum and maximum delay constraints, you can set the output delay for the output port with a value of zero. You can use an existing clock from the design or a virtual clock as the clock reference.

Related Links

- set_max_delay
- set_min_delay
  For more information about these commands, refer to Quartus Prime Help.

3.3.5.4 Delay Annotation

To modify the default delay values used during timing analysis, use the `set_annotated_delay` and `set_timing_derate` commands. You must update the timing netlist to see the results of these commands

To specify different operating conditions in a single SDC file, rather than having multiple SDC files that specify different operating conditions, use the `set_annotated_delay -operating_conditions` command.

Related Links

- set_timing_derate
- set_annotated_delay
  For more information about these commands, refer to the Quartus Prime Help.

3.3.5.5 Multicycle Paths

By default, the TimeQuest analyzer performs a single-cycle analysis, which is the most restrictive type of analysis. When analyzing a path, the setup launch and latch edge times are determined by finding the closest two active edges in the respective waveforms.

For a hold analysis, the timing analyzer analyzes the path against two timing conditions for every possible setup relationship, not just the worst-case setup relationship. Therefore, the hold launch and latch times may be completely unrelated to the setup launch and latch edges. The TimeQuest analyzer does not report negative

setup or hold relationships. When either a negative setup or a negative hold relationship is calculated, the TimeQuest analyzer moves both the launch and latch edges such that the setup and hold relationship becomes positive.

A multicycle constraint adjusts setup or hold relationships by the specified number of clock cycles based on the source (`-start`) or destination (`-end`) clock. An end setup multicycle constraint of 2 extends the worst-case setup latch edge by one destination clock period. If `-start` and `-end` values are not specified, the default constraint is `-end`.

Hold multicycle constraints are based on the default hold position (the default value is 0). An end hold multicycle constraint of 1 effectively subtracts one destination clock period from the default hold latch edge.

When the objects are timing nodes, the multicycle constraint only applies to the path between the two nodes. When an object is a clock, the multicycle constraint applies to all paths where the source node (`-from`) or destination node (`-to`) is clocked by the clock. When you adjust a setup relationship with a multicycle constraint, the hold relationship is adjusted automatically.

You can use TimeQuest analyzer commands to modify either the launch or latch edge times that the uses to determine a setup relationship or hold relationship.

Table 17. Commands to Modify Edge Times

Command	Description of Modification
`set_multicycle_path -setup -end <value>`	Latch edge time of the setup relationship
`set_multicycle_path -setup -start<value>`	Launch edge time of the setup relationship
`set_multicycle_path -hold -end <value>`	Latch edge time of the hold relationship
`set_multicycle_path -hold -start <value>`	Launch edge time of the hold relationship

3.3.5.6 Common Multicycle Variations

Multicycle exceptions adjust the timing requirements for a register-to-register path, allowing the Fitter to optimally place and route a design in a device. Multicycle exceptions also can reduce compilation time and improve the quality of results, and can be used to change timing requirements. Two common multicycle variations are relaxing setup to allow a slower data transfer rate, and altering the setup to account for a phase shift.

3.3.5.6.1 Relaxing Setup with set_multicyle_path

A common type of multicycle exception occurs when the data transfer rate is slower than the clock cycle. Relaxing the setup relationship opens the window when data is accepted as valid.

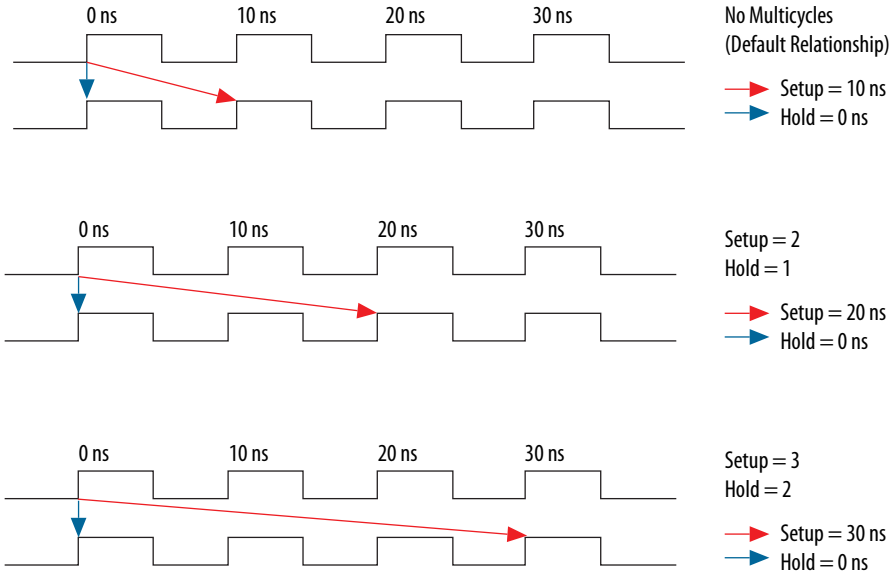
In this example, the source clock has a period of 10 ns, but a group of registers are enabled by a toggling clock, so they only toggle every other cycle. Since they are fed by a 10 ns clock, the TimeQuest analyzer reports a set up of 10 ns and a hold of 0 ns, However, since the data is transferring every other cycle, the relationships should be analyzed as if the clock were operating at 20 ns, which would result in a setup of 20 ns, while the hold remains 0 ns, in essence, extending the window of time when the data can be recognized.

The following pair of multicycle assignments relax the setup relationship by specifying the `-setup` value of N and the `-hold` value as N-1. You must specify the hold relationship with a `-hold` assignment to prevent a positive hold requirement.

Relaxing Setup while Maintaining Hold

```
set_multicycle_path -setup -from src_reg* -to dst_reg* 2
set_multicycle_path -hold -from src_reg* -to dst_reg* 1
```

Figure 50. Relaxing Setup by Multiple Cycles



This pattern can be extended to create larger setup relationships in order to ease timing closure requirements. A common use for this exception is when writing to asynchronous RAM across an I/O interface. The delay between address, data, and a write enable may be several cycles. A multicycle exception to I/O ports can allow extra time for the address and data to resolve before the enable occurs.

You can relax the setup by three cycles with the following code in your SDC file.

Three Cycle I/O Interface Exception

```
set_multicycle_path -setup -to [get_ports {SRAM_ADD[*] SRAM_DATA[*]} 3
set_multicycle_path -hold -to [get_ports {SRAM_ADD[*] SRAM_DATA[*]} 2
```

3.3.5.6.2 Accounting for a Phase Shift

In this example, the design contains a PLL that performs a phase-shift on a clock whose domain exchanges data with domains that do not experience the phase shift. A common example is when the destination clock is phase-shifted forward and the source clock is not, the default setup relationship becomes that phase-shift, thus shifting the window when data is accepted as valid.

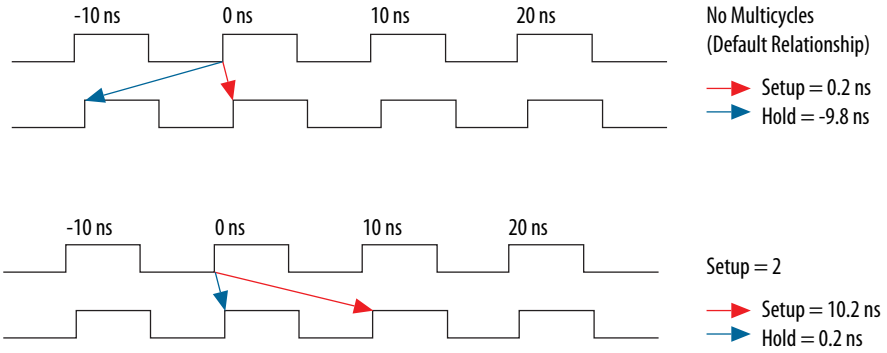
For example, the following code is a circumstance where a PLL phase-shifts one output forward by a small amount, in this case 0.2 ns.

Cross Domain Phase-Shift

```
create_generated_clock -source pll|inclk[0] -name pll|clk[0] pll|clk[0]
create_generated_clock -source pll|inclk[0] -name pll|clk[1] -phase 30 pll|clk[1]
```

The default setup relationship for this phase-shift is 0.2 ns, shown in Figure A, creating a scenario where the hold relationship is negative, which makes achieving timing closure nearly impossible.

Figure 51. Phase-Shifted Setup and Hold



Adding the following constraint in your SDC allows the data to transfer to the following edge.

```
set_multicycle_path -setup -from [get_clocks clk_a] -to [get_clocks clk_b] 2
```

The hold relationship is derived from the setup relationship, making a multicyle hold constraint unnecessary.

Related Links

- Same Frequency Clocks with Destination Clock Offset on page 84
  Refer to this topic for a more complete example.
- set_multicycle_path
  For more information about this command, refer to the Quartus Prime Help.

3.3.5.7 Examples of Basic Multicycle Exceptions

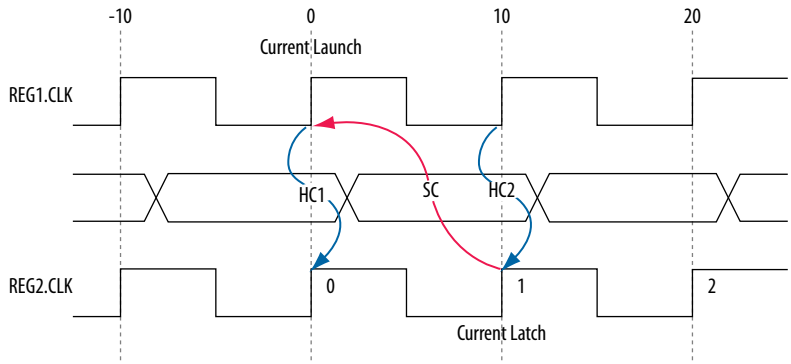
Each example explains how the multicycle exceptions affect the default setup and hold analysis in the TimeQuest analyzer. The multicycle exceptions are applied to a simple register-to-register circuit. Both the source and destination clocks are set to 10 ns.

3.3.5.7.1 Default Settings

By default, the TimeQuest analyzer performs a single-cycle analysis to determine the setup and hold checks. Also, by default, the TimeQuest analyzer sets the end multicycle setup assignment value to one and the end multicycle hold assignment value to zero.

The source and the destination timing waveform for the source register and destination register, respectively where HC1 and HC2 are hold checks one and two and SC is the setup check.

Figure 52. Default Timing Diagram



The calculation that the TimeQuest analyzer performs to determine the setup check.

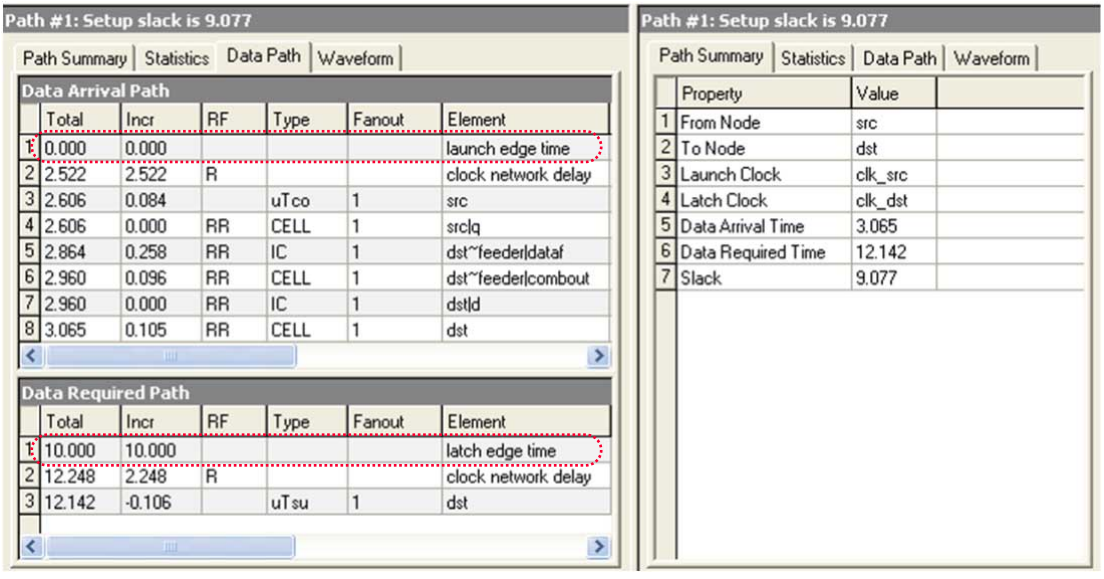
Figure 53. Setup Check

$$\begin{aligned}\text{setup check} &= \text{current latch edge} - \text{closest previous launch edge}\\ &= 10\text{ ns} - 0\text{ ns}\\ &= 10\text{ ns}\end{aligned}$$

The most restrictive setup relationship with the default single-cycle analysis, that is, a setup relationship with an end multicycle setup assignment of one, is 10 ns.

The setup report for the default setup in the TimeQuest analyzer with the launch and latch edges highlighted.

Figure 54. Setup Report



Path #1: Setup slack is 9.077

Path Summary | Statistics | Data Path | Waveform

Data Arrival Path

	Total	Incr	RF	Type	Fanout	Element
1	0.000	0.000				launch edge time
2	2.522	2.522	R			clock network delay
3	2.606	0.084		uTco	1	src
4	2.606	0.000	RR	CELL	1	srclq
5	2.864	0.258	RR	IC	1	dst~feeder|dataf
6	2.960	0.096	RR	CELL	1	dst~feeder|combout
7	2.960	0.000	RR	IC	1	dst|d
8	3.065	0.105	RR	CELL	1	dst

Data Required Path

	Total	Incr	RF	Type	Fanout	Element
1	10.000	10.000				latch edge time
2	12.248	2.248	R			clock network delay
3	12.142	-0.106		uTsu	1	dst

Path #1: Setup slack is 9.077

Path Summary | Statistics | Data Path | Waveform

	Property	Value
1	From Node	src
2	To Node	dst
3	Launch Clock	clk_src
4	Latch Clock	clk_dst
5	Data Arrival Time	3.065
6	Data Required Time	12.142
7	Slack	9.077

The calculation that the TimeQuest analyzer performs to determine the hold check. Both hold checks are equivalent.

Figure 55. Hold Check

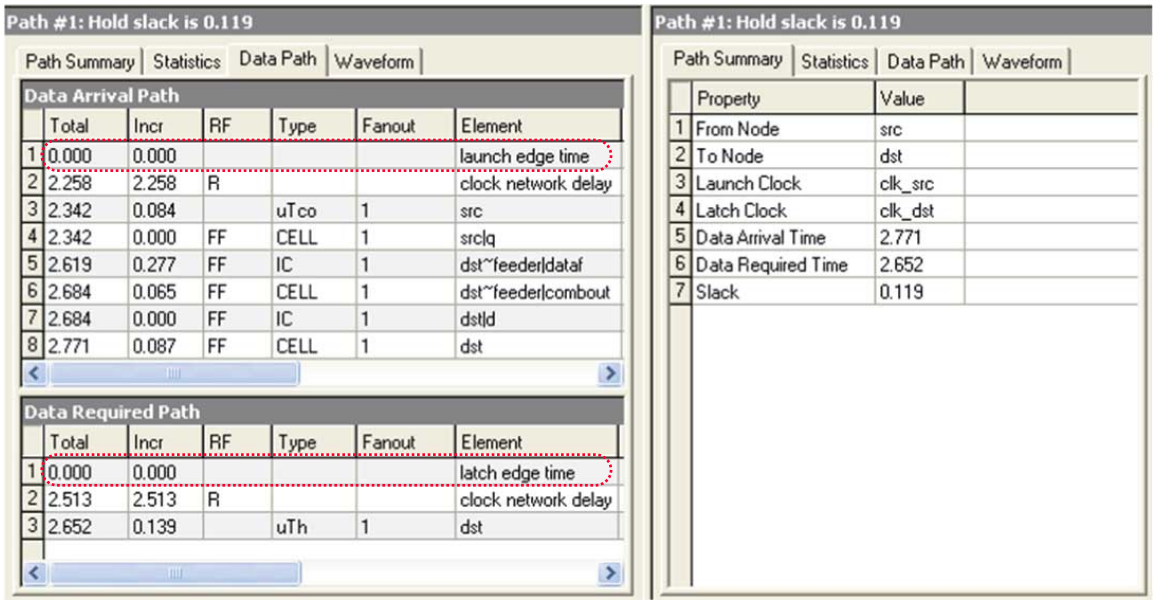
$$\begin{aligned} \text{hold check 1} &= \text{current launch edge} - \text{previous latch edge} \\ &= 0\text{ ns} - 0\text{ ns} \\ &= 0\text{ ns} \end{aligned}$$

$$\begin{aligned} \text{hold check 2} &= \text{next launch edge} - \text{current latch edge} \\ &= 10\text{ ns} - 10\text{ ns} \\ &= 0\text{ ns} \end{aligned}$$

The most restrictive hold relationship with the default single-cycle analysis, that a hold relationship with an end multicycle hold assignment of zero, is 0 ns.

The hold report for the default setup in the TimeQuest analyzer with the launch and latch edges highlighted.

Figure 56. Hold Report



3.3.5.7.2 End Multicycle Setup = 2 and End Multicycle Hold = 0

In this example, the end multicycle setup assignment value is two, and the end multicycle hold assignment value is zero.

Multicycle Exceptions

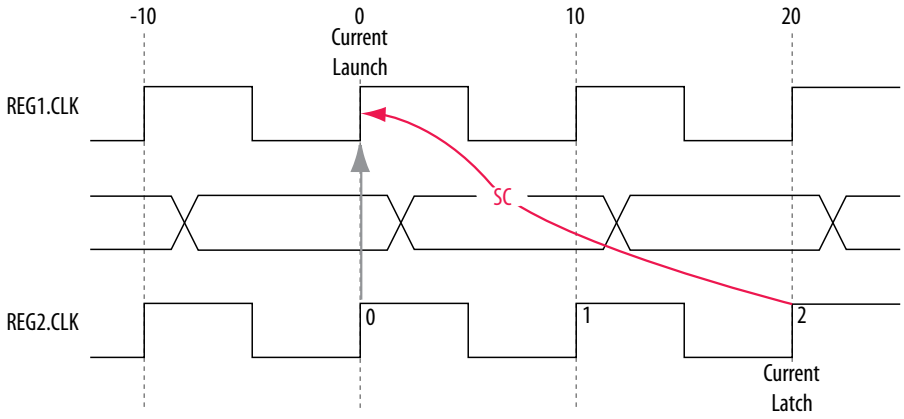
```
set_multicycle_path -from [get_clocks clk_src] -to [get_clocks clk_dst] \
    -setup -end 2
```

Note: An end multicycle hold value is not required because the default end multicycle hold value is zero.

In this example, the setup relationship is relaxed by a full clock period by moving the latch edge to the next latch edge. The hold analysis is unchanged from the default settings.

The setup timing diagram for the analysis that the TimeQuest analyzer performs. The latch edge is a clock cycle later than in the default single-cycle analysis.

Figure 57. Setup Timing Diagram



The calculation that the TimeQuest analyzer performs to determine the setup check.

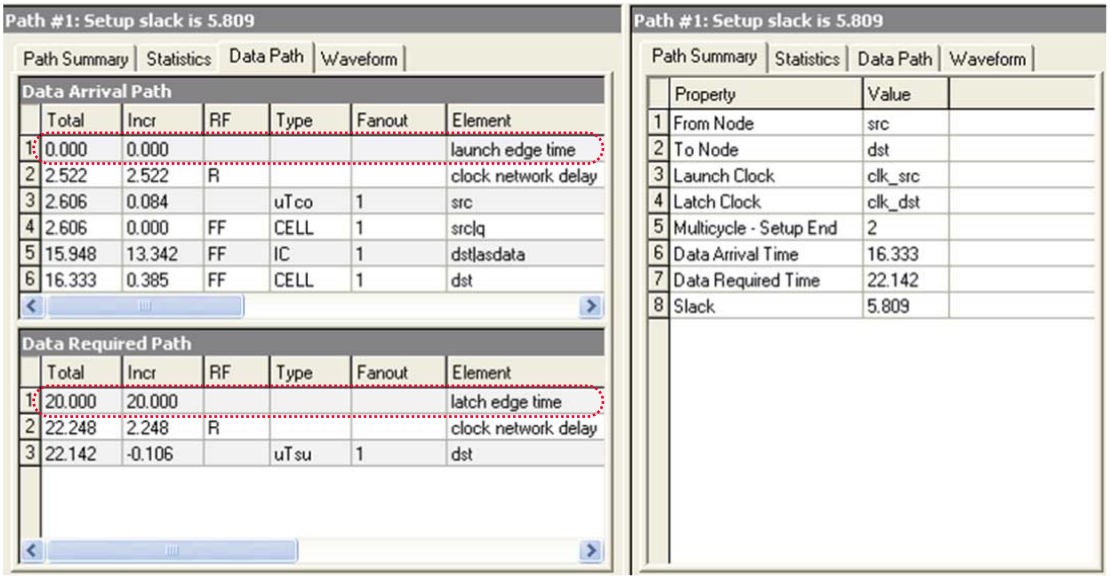
Figure 58. Setup Check

$$\begin{aligned} \text{setup check} &= \text{current latch edge} - \text{closest previous launch edge} \\ &= 20\ \text{ns} - 0\ \text{ns} \\ &= 20\ \text{ns} \end{aligned}$$

The most restrictive setup relationship with an end multicycle setup assignment of two is 20 ns.

The setup report in the TimeQuest analyzer with the launch and latch edges highlighted.

Figure 59. Setup Report



Path #1: Setup slack is 5.809

Path Summary | Statistics | Data Path | Waveform

Data Arrival Path

	Total	Incr	RF	Type	Fanout	Element
1	0.000	0.000				launch edge time
2	2.522	2.522	R			clock network delay
3	2.606	0.084		uTco	1	src
4	2.606	0.000	FF	CELL	1	srclq
5	15.948	13.342	FF	IC	1	dstlasdata
6	16.333	0.385	FF	CELL	1	dst

Data Required Path

	Total	Incr	RF	Type	Fanout	Element
1	20.000	20.000				latch edge time
2	22.248	2.248	R			clock network delay
3	22.142	-0.106		uTsu	1	dst

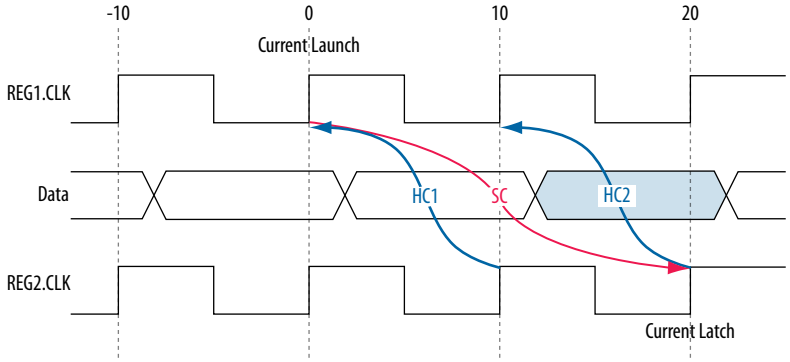
Path #1: Setup slack is 5.809

Path Summary | Statistics | Data Path | Waveform

	Property	Value
1	From Node	src
2	To Node	dst
3	Launch Clock	clk_src
4	Latch Clock	clk_dst
5	Multicycle - Setup End	2
6	Data Arrival Time	16.333
7	Data Required Time	22.142
8	Slack	5.809

Because the multicycle hold latch and launch edges are the same as the results of hold analysis with the default settings, the multicycle hold analysis in this example is equivalent to the single-cycle hold analysis. The hold checks are relative to the setup check. Usually, the TimeQuest analyzer performs hold checks on every possible setup check, not only on the most restrictive setup check edges.

Figure 60. Hold Timing DIagram



The calculation that the TimeQuest analyzer performs to determine the hold check. Both hold checks are equivalent.

Figure 61.

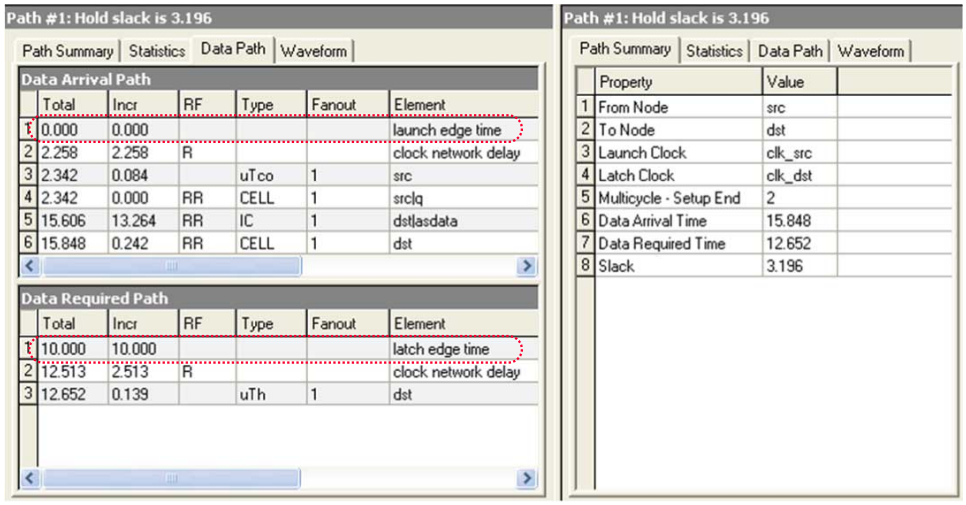
$$\text{hold check 1} = \text{current launch edge} - \text{previous latch edge} = 0\text{ ns} - 10\text{ ns} = -10\text{ ns}$$

$$\text{hold check 2} = \text{next launch edge} - \text{current latch edge} = 10\text{ ns} - 20\text{ ns} = -10\text{ ns}$$

The most restrictive hold relationship with an end multicycle setup assignment value of two and an end multicycle hold assignment value of zero is 10 ns.

The hold report for this example in the TimeQuest analyzer with the launch and latch edges highlighted.

Figure 62. Hold Report



3.3.5.7.3 End Multicycle Setup = 2 and End Multicycle Hold = 1

In this example, the end multicycle setup assignment value is two, and the end multicycle hold assignment value is one.

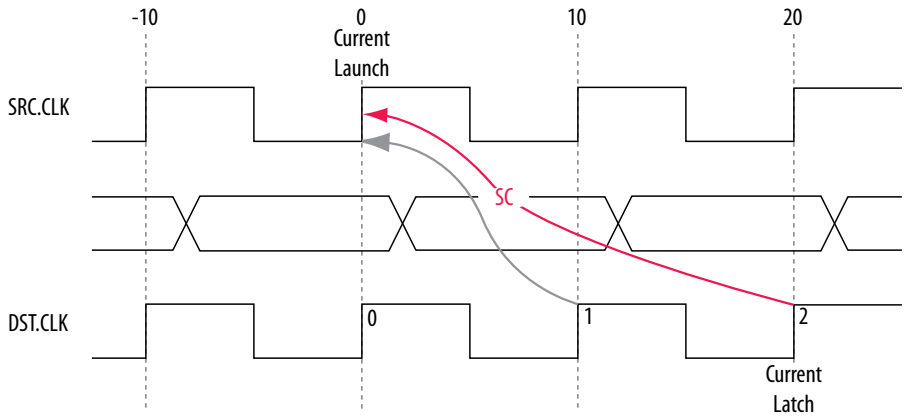
Multicycle Exceptions

```
set_multicycle_path -from [get_clocks clk_src] -to [get_clocks clk_dst] \
    -setup -end 2
set_multicycle_path -from [get_clocks clk_src] -to [get_clocks clk_dst] -hold -
end 1
```

In this example, the setup relationship is relaxed by two clock periods by moving the latch edge to the left two clock periods. The hold relationship is relaxed by a full period by moving the latch edge to the previous latch edge.

The setup timing diagram for the analysis that the TimeQuest analyzer performs.

Figure 63. Setup Timing Diagram



The calculation that the TimeQuest analyzer performs to determine the setup check.

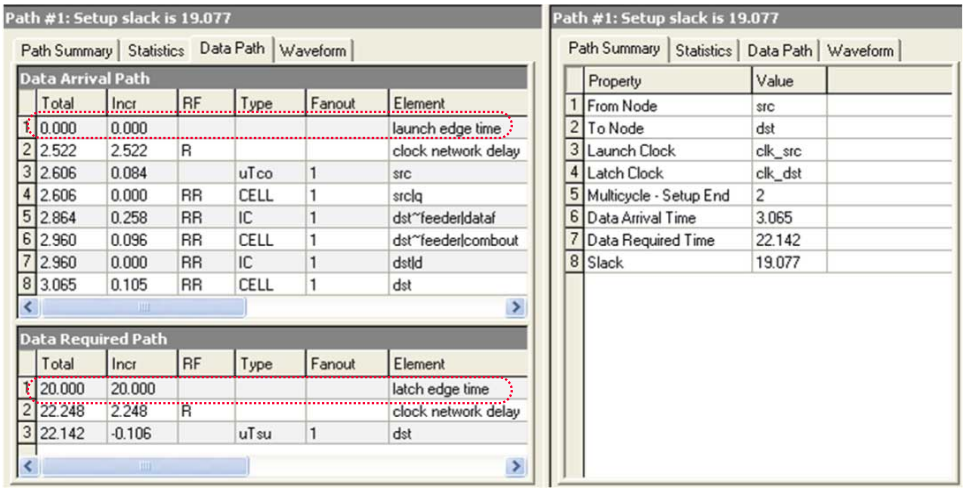
Figure 64. Setup Check

$$\begin{aligned} \text{setup check} &= \text{current latch edge} - \text{closest previous launch edge} \\ &= 20\ \text{ns} - 0\ \text{ns} \\ &= 20\ \text{ns} \end{aligned}$$

The most restrictive hold relationship with an end multicycle setup assignment value of two is 20 ns.

The setup report for this example in the TimeQuest analyzer with the launch and latch edges highlighted.

Figure 65. Setup Report



Path #1: Setup slack is 19.077

Path Summary | Statistics | Data Path | Waveform

Data Arrival Path

	Total	Incr	RF	Type	Fanout	Element
1	0.000	0.000				launch edge time
2	2.522	2.522	R			clock network delay
3	2.606	0.084		uTco	1	src
4	2.606	0.000	RR	CELL	1	srclq
5	2.864	0.258	RR	IC	1	dst~feeder|dataf
6	2.960	0.096	RR	CELL	1	dst~feeder|combout
7	2.960	0.000	RR	IC	1	dst|d
8	3.065	0.105	RR	CELL	1	dst

Data Required Path

	Total	Incr	RF	Type	Fanout	Element
1	20.000	20.000				latch edge time
2	22.248	2.248	R			clock network delay
3	22.142	-0.106		uTsu	1	dst

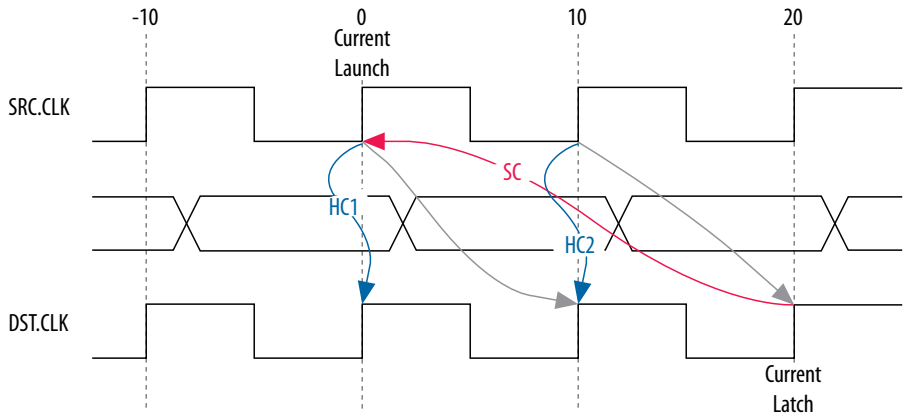
Path #1: Setup slack is 19.077

Path Summary | Statistics | Data Path | Waveform

	Property	Value
1	From Node	src
2	To Node	dst
3	Launch Clock	clk_src
4	Latch Clock	clk_dst
5	Multicycle - Setup End	2
6	Data Arrival Time	3.065
7	Data Required Time	22.142
8	Slack	19.077

The timing diagram for the hold checks for this example. The hold checks are relative to the setup check.

Figure 66. Hold Timing Diagram



The calculation that the TimeQuest analyzer performs to determine the hold check. Both hold checks are equivalent.

Figure 67. Hold Check

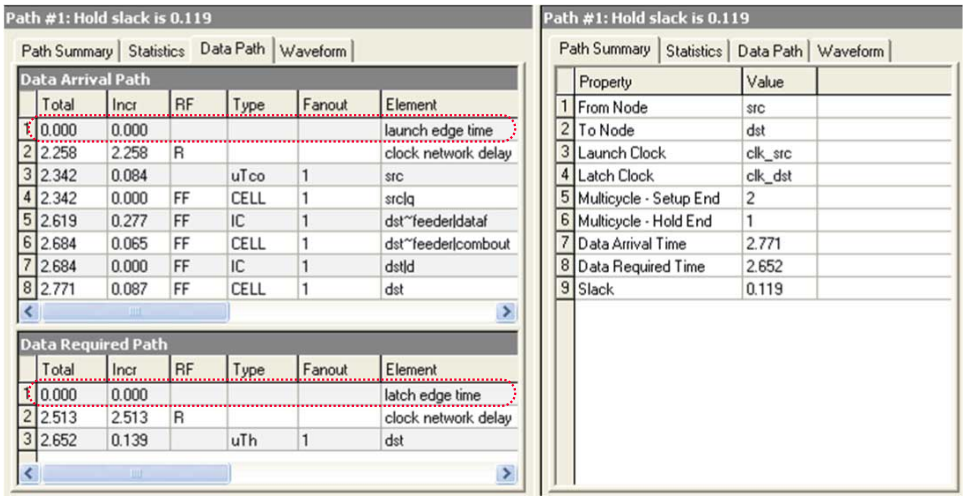
$$\begin{aligned} \text{hold check 1} &= \text{current launch edge} - \text{previous latch edge} \\ &= 0\ \text{ns} - 0\ \text{ns} \\ &= 0\ \text{ns} \end{aligned}$$

$$\begin{aligned} \text{hold check 2} &= \text{next launch edge} - \text{current latch edge} \\ &= 10\ \text{ns} - 10\ \text{ns} \\ &= 0\ \text{ns} \end{aligned}$$

The most restrictive hold relationship with an end multicycle setup assignment value of two and an end multicycle hold assignment value of one is 0 ns.

The hold report for this example in the TimeQuest analyzer with the launch and latch edges highlighted.

Figure 68. Hold Report



3.3.5.8 Application of Multicycle Exceptions

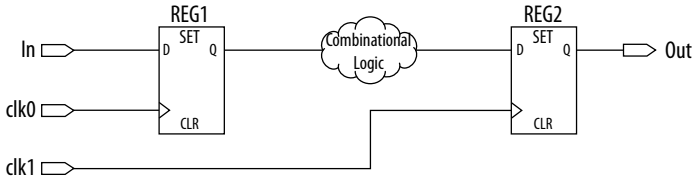
This section shows the following examples of applications of multicycle exceptions. Each example explains how the multicycle exceptions affect the default setup and hold analysis in the TimeQuest analyzer. All of the examples are between related clock domains. If your design contains related clocks, such as PLL clocks, and paths between related clock domains, you can apply multicycle constraints.

3.3.5.8.1 Same Frequency Clocks with Destination Clock Offset

In this example, the source and destination clocks have the same frequency, but the destination clock is offset with a positive phase shift. Both the source and destination clocks have a period of 10 ns. The destination clock has a positive phase shift of 2 ns with respect to the source clock.

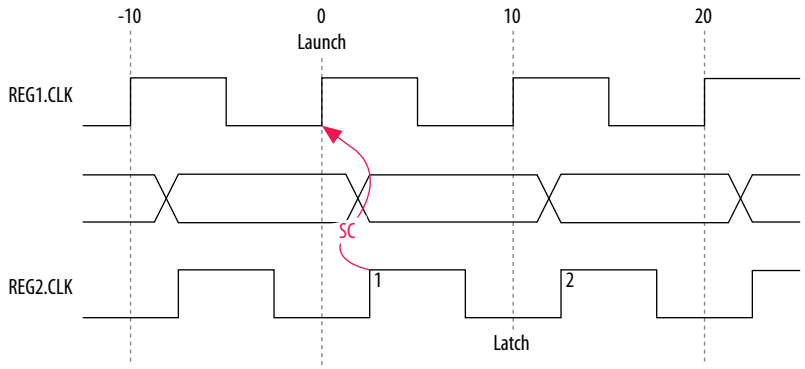
An example of a design with same frequency clocks and a destination clock offset.

Figure 69. Same Frequency Clocks with Destination Clock Offset



The timing diagram for default setup check analysis that the TimeQuest analyzer performs.

Figure 70. Setup Timing Diagram



The calculation that the TimeQuest analyzer performs to determine the setup check.

Figure 71. Setup Check

$$\begin{aligned} \text{setup check} &= \text{current latch edge} - \text{closest previous launch edge} \\ &= 2\text{ ns} - 0\text{ ns} \\ &= 2\text{ ns} \end{aligned}$$

The setup relationship shown is too pessimistic and is not the setup relationship required for typical designs. To correct the default analysis, you must use an end multicycle setup exception of two. A multicycle exception used to correct the default analysis in this example in your SDC file.

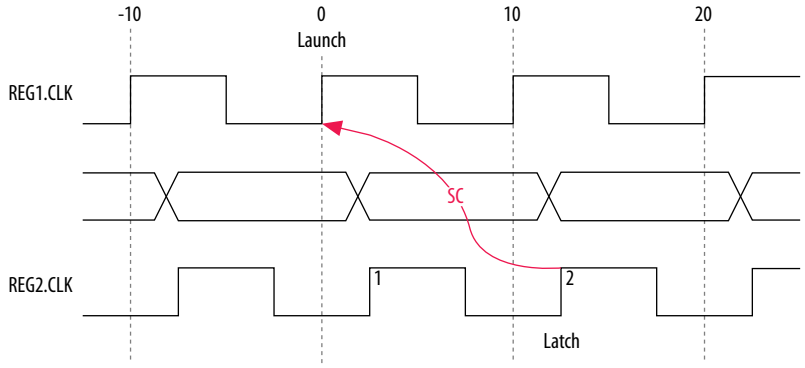
Multicycle Exceptions

```
set_multicycle_path -from [get_clocks clk_src] -to [get_clocks clk_dst] \
     -setup -end 2
```

The timing diagram for the preferred setup relationship for this example.

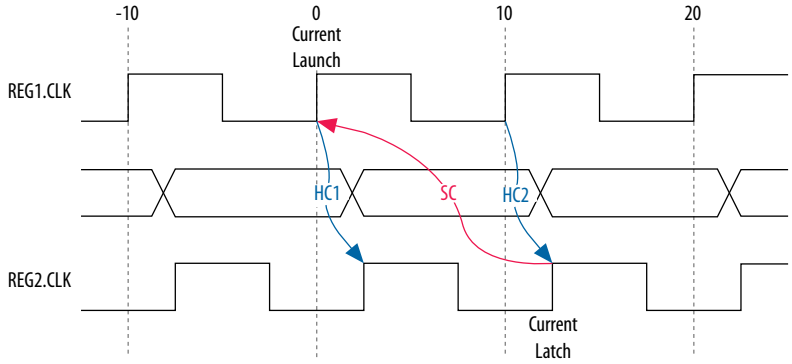
.

Figure 72. Preferred Setup Relationship



The timing diagram for default hold check analysis that the TimeQuest analyzer performs with an end multicycle setup value of two.

Figure 73. Default Hold Check



The calculation that the TimeQuest analyzer performs to determine the hold check.

Figure 74. Hold Check

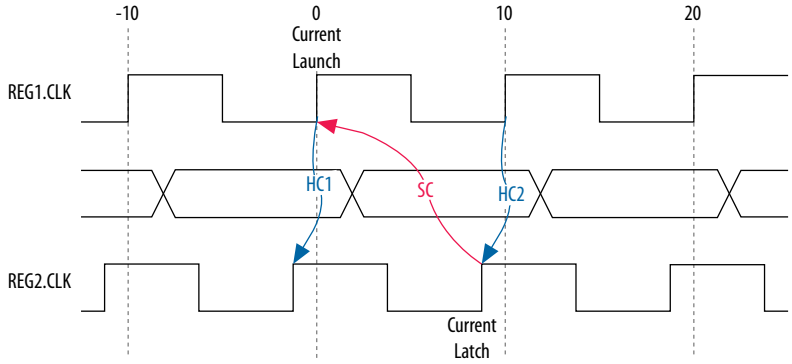
$$\begin{aligned} \text{hold check 1} &= \text{current launch edge} - \text{previous latch edge} \\ &= 0\text{ ns} - 2\text{ ns} \\ &= -2\text{ ns} \end{aligned}$$

$$\begin{aligned} \text{hold check 2} &= \text{next launch edge} - \text{current latch edge} \\ &= 10\text{ ns} - 12\text{ ns} \\ &= -2\text{ ns} \end{aligned}$$

In this example, the default hold analysis returns the preferred hold requirements and no multicycle hold exceptions are required.

The associated setup and hold analysis if the phase shift is –2 ns. In this example, the default hold analysis is correct for the negative phase shift of 2 ns, and no multicycle exceptions are required.

Figure 75. Negative Phase Shift

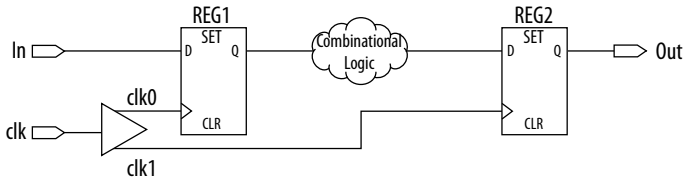


3.3.5.8.2 Destination Clock Frequency is a Multiple of the Source Clock Frequency

In this example, the destination clock frequency value of 5 ns is an integer multiple of the source clock frequency of 10 ns. The destination clock frequency can be an integer multiple of the source clock frequency when a PLL is used to generate both clocks with a phase shift applied to the destination clock.

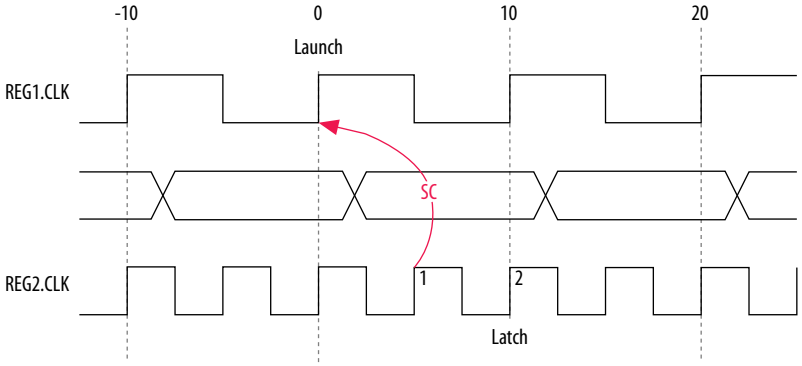
An example of a design where the destination clock frequency is a multiple of the source clock frequency.

Figure 76. Destination Clock is Multiple of Source Clock



The timing diagram for default setup check analysis that the TimeQuest analyzer performs.

Figure 77. Setup Timing Diagram



The calculation that the TimeQuest analyzer performs to determine the setup check.

Figure 78. Setup Check

$$\begin{aligned} \text{setup check} &= \text{current latch edge} - \text{closest previous launch edge} \\ &= 5\text{ ns} - 0\text{ ns} \\ &= 5\text{ ns} \end{aligned}$$

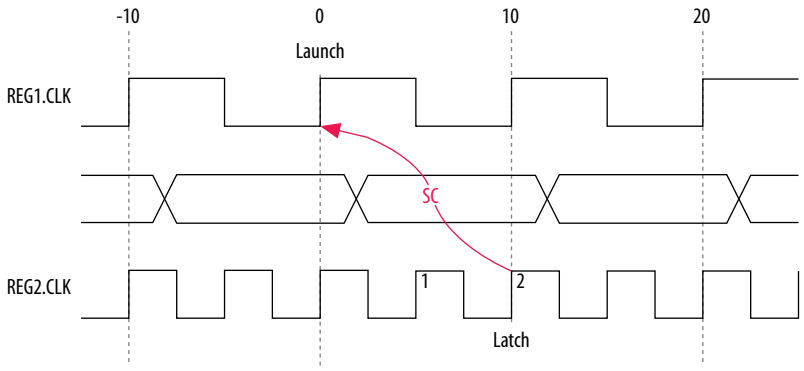
The setup relationship demonstrates that the data does not need to be captured at edge one, but can be captured at edge two; therefore, you can relax the setup requirement. To correct the default analysis, you must shift the latch edge by one clock period with an end multicycle setup exception of two. The multicycle exception assignment used to correct the default analysis in this example.

Multicycle Exceptions

```
set_multicycle_path -from [get_clocks clk_src] -to [get_clocks clk_dst] \
     -setup -end 2
```

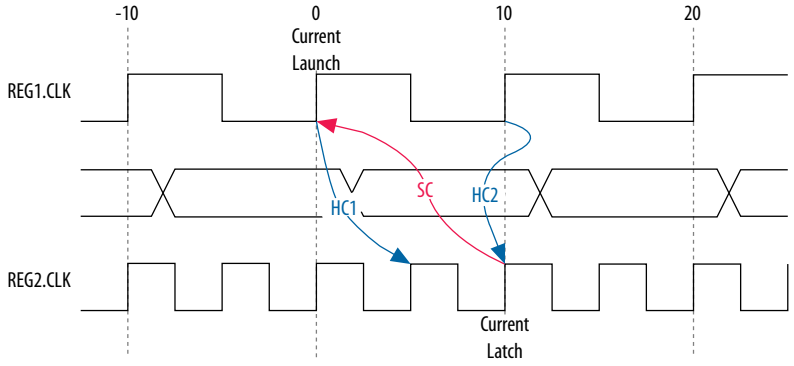
The timing diagram for the preferred setup relationship for this example.

Figure 79. Preferred Setup Analysis



The timing diagram for default hold check analysis performed by the TimeQuest analyzer with an end multicycle setup value of two.

Figure 80. Default Hold Check



The calculation that the TimeQuest analyzer performs to determine the hold check.

Figure 81. Hold Check

$$\begin{aligned} \text{hold check 1} &= \text{current launch edge} - \text{previous latch edge} \\ &= 0\text{ ns} - 5\text{ ns} \\ &= -5\text{ ns} \end{aligned}$$

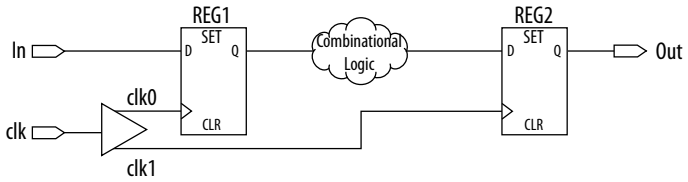
$$\begin{aligned} \text{hold check 2} &= \text{next launch edge} - \text{current latch edge} \\ &= 10\text{ ns} - 10\text{ ns} \\ &= 0\text{ ns} \end{aligned}$$

In this example, hold check one is too restrictive. The data is launched by the edge at 0 ns and should check against the data captured by the previous latch edge at 0 ns, which does not occur in hold check one. To correct the default analysis, you must use an end multicycle hold exception of one.

3.3.5.8.3 Destination Clock Frequency is a Multiple of the Source Clock Frequency with an Offset

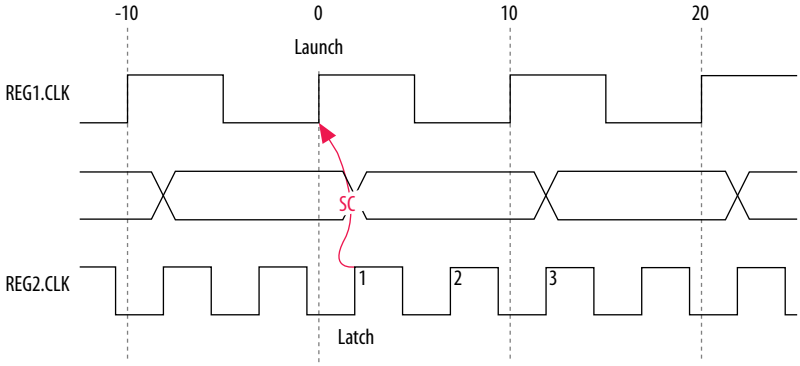
This example is a combination of the previous two examples. The destination clock frequency is an integer multiple of the source clock frequency and the destination clock has a positive phase shift. The destination clock frequency is 5 ns and the source clock frequency is 10 ns. The destination clock also has a positive offset of 2 ns with respect to the source clock. The destination clock frequency can be an integer multiple of the source clock frequency with an offset when a PLL is used to generate both clocks with a phase shift applied to the destination clock. The following example shows a design in which the destination clock frequency is a multiple of the source clock frequency with an offset.

Figure 82. Destination Clock is Multiple of Source Clock with Offset



The timing diagram for the default setup check analysis the TimeQuest analyzer performs.

Figure 83. Setup Timing Diagram



The calculation that the TimeQuest analyzer performs to determine the setup check.

Figure 84. Hold Check

$$\begin{aligned} \text{setup check} &= \text{current latch edge} - \text{closest previous launch edge} \\ &= 2\ \text{ns} - 0\ \text{ns} \\ &= 2\ \text{ns} \end{aligned}$$

The setup relationship in this example demonstrates that the data does not need to be captured at edge one, but can be captured at edge two; therefore, you can relax the setup requirement. To correct the default analysis, you must shift the latch edge by one clock period with an end multicycle setup exception of three.

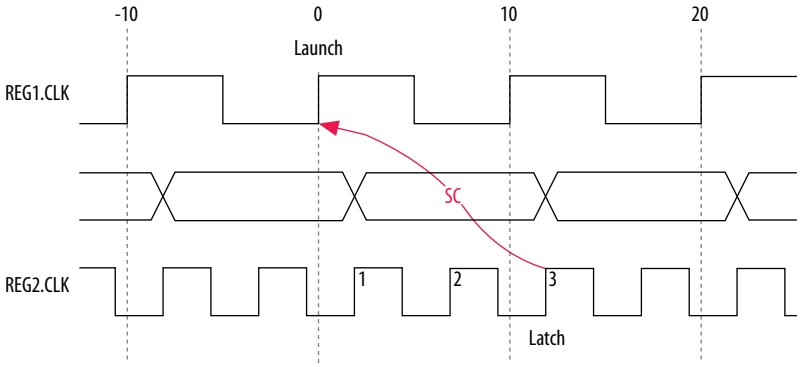
The multicycle exception code you can use to correct the default analysis in this example.

Multicycle Exceptions

```
set_multicycle_path -from [get_clocks clk_src] -to [get_clocks clk_dst] \
     -setup -end 3
```

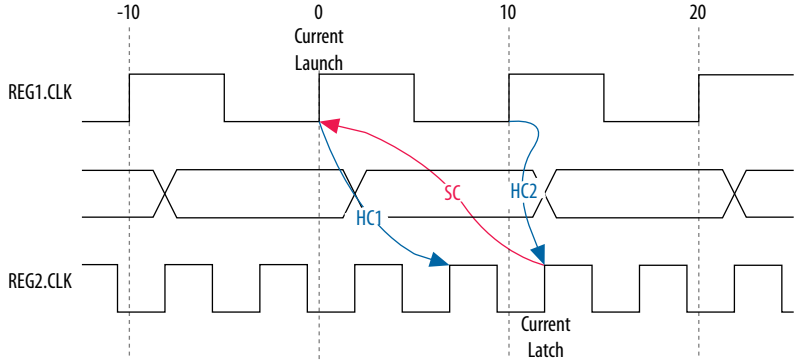
The timing diagram for the preferred setup relationship for this example.

Figure 85. Preferred Setup Analysis



The timing diagram for default hold check analysis the TimeQuest analyzer performs with an end multicycle setup value of three.

Figure 86. Default Hold Check



The calculation that the TimeQuest analyzer performs to determine the hold check.

Figure 87. Hold Check

$$\begin{aligned} \text{hold check 1} &= \text{current launch edge} - \text{previous latch edge} \\ &= 0\text{ ns} - 5\text{ ns} \\ &= -5\text{ ns} \end{aligned}$$

$$\begin{aligned} \text{hold check 2} &= \text{next launch edge} - \text{current latch edge} \\ &= 10\text{ ns} - 10\text{ ns} \\ &= 0\text{ ns} \end{aligned}$$

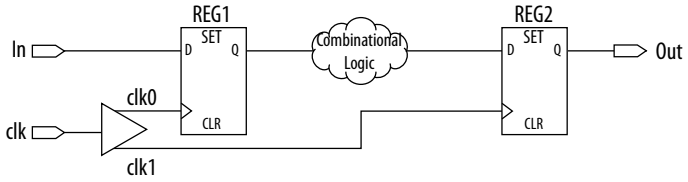
In this example, hold check one is too restrictive. The data is launched by the edge at 0 ns and should check against the data captured by the previous latch edge at 2 ns, which does not occur in hold check one. To correct the default analysis, you must use an end multicycle hold exception of one.

3.3.5.8.4 Source Clock Frequency is a Multiple of the Destination Clock Frequency

In this example, the source clock frequency value of 5 ns is an integer multiple of the destination clock frequency of 10 ns. The source clock frequency can be an integer multiple of the destination clock frequency when a PLL is used to generate both clocks and different multiplication and division factors are used.

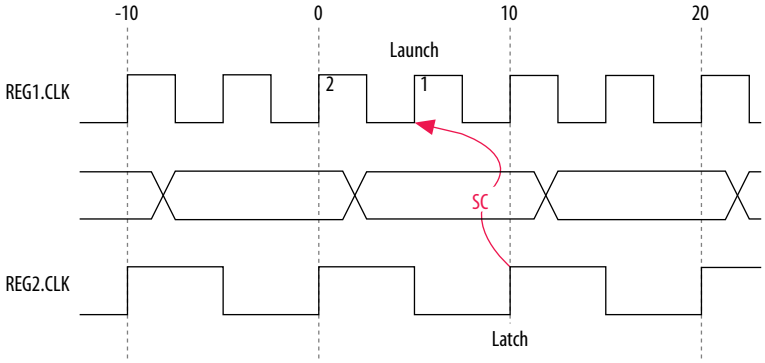
An example of a design where the source clock frequency is a multiple of the destination clock frequency.

Figure 88. Source Clock Frequency is Multiple of Destination Clock Frequency



The timing diagram for default setup check analysis performed by the TimeQuest analyzer.

Figure 89. Default Setup Check Analysis



The calculation that the TimeQuest analyzer performs to determine the setup check.

Figure 90. Setup Check

$$\begin{aligned} \text{setup check} &= \text{current latch edge} - \text{closest previous launch edge} \\ &= 10\text{ ns} - 5\text{ ns} \\ &= 5\text{ ns} \end{aligned}$$

The setup relationship shown demonstrates that the data launched at edge one does not need to be captured, and the data launched at edge two must be captured; therefore, you can relax the setup requirement. To correct the default analysis, you must shift the launch edge by one clock period with a start multicycle setup exception of two.

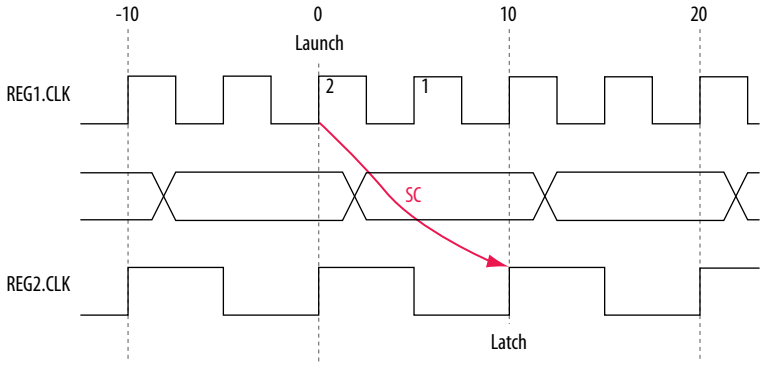
The multicycle exception code you can use to correct the default analysis in this example.

Multicycle Exceptions

```
set_multicycle_path -from [get_clocks clk_src] -to [get_clocks clk_dst] \
     -setup -start 2
```

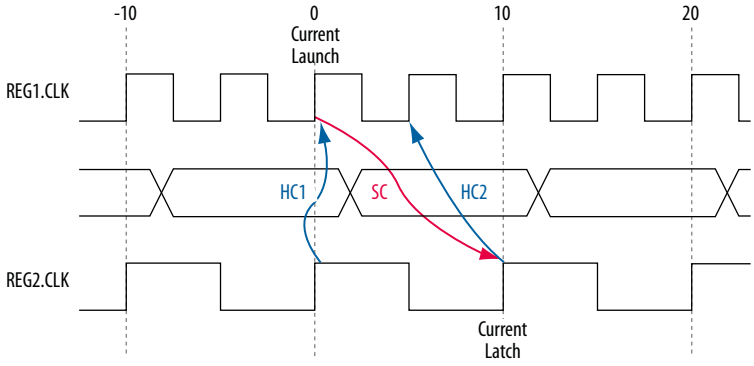
The timing diagram for the preferred setup relationship for this example.

Figure 91. Preferred Setup Check Analysis



The timing diagram for default hold check analysis the TimeQuest analyzer performs for a start multicycle setup value of two.

Figure 92. Default Hold Check



The calculation that the TimeQuest analyzer performs to determine the hold check.

Figure 93. Hold Check

$$\begin{aligned} \text{hold check 1} &= \text{current launch edge} - \text{previous latch edge} \\ &= 0\ \text{ns} - 0\ \text{ns} \\ &= 0\ \text{ns} \end{aligned}$$

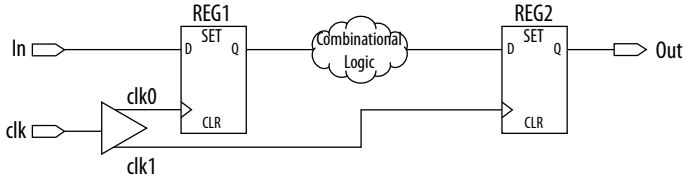
$$\begin{aligned} \text{hold check 2} &= \text{next launch edge} - \text{current latch edge} \\ &= 5\ \text{ns} - 10\ \text{ns} \\ &= -5\ \text{ns} \end{aligned}$$

In this example, hold check two is too restrictive. The data is launched next by the edge at 10 ns and should check against the data captured by the current latch edge at 10 ns, which does not occur in hold check two. To correct the default analysis, you must use a start multicycle hold exception of one.

3.3.5.8.5 Source Clock Frequency is a Multiple of the Destination Clock Frequency with an Offset

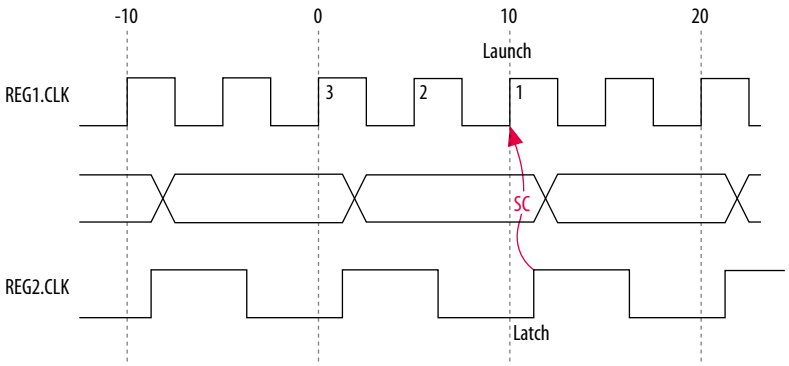
In this example, the source clock frequency is an integer multiple of the destination clock frequency and the destination clock has a positive phase offset. The source clock frequency is 5 ns and destination clock frequency is 10 ns. The destination clock also has a positive offset of 2 ns with respect to the source clock. The source clock frequency can be an integer multiple of the destination clock frequency with an offset when a PLL is used to generate both clocks, different multiplication.

Figure 94. Source Clock Frequency is Multiple of Destination Clock Frequency with Offset



Timing diagram for default setup check analysis the TimeQuest analyzer performs.

Figure 95. Setup Timing Diagram



The calculation that the TimeQuest analyzer performs to determine the setup check.

Figure 96. Setup Check

$$\begin{aligned} \text{setup check} &= \text{current latch edge} - \text{closest previous launch edge} \\ &= 12\ \text{ns} - 10\ \text{ns} \\ &= 2\ \text{ns} \end{aligned}$$

The setup relationship in this example demonstrates that the data is not launched at edge one, and the data that is launched at edge three must be captured; therefore, you can relax the setup requirement. To correct the default analysis, you must shift the launch edge by two clock periods with a start multicycle setup exception of three.

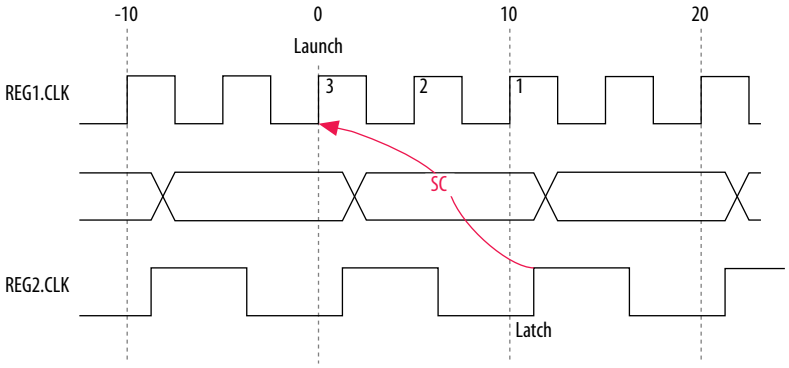
The multicycle exception used to correct the default analysis in this example.

Multicycle Exceptions

```
set_multicycle_path -from [get_clocks clk_src] -to [get_clocks clk_dst] \
     -setup -start 3
```

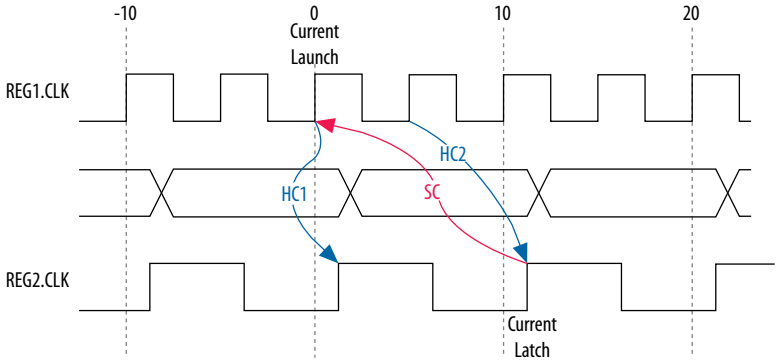
The timing diagram for the preferred setup relationship for this example.

Figure 97. Preferred Setup Check Analysis



The timing diagram for default hold check analysis the TimeQuest analyzer performs for a start multicycle setup value of three.

Figure 98. Default Hold Check Analysis



The calculation that the TimeQuest analyzer performs to determine the hold check.

Figure 99. Hold Check

$$\begin{aligned} \text{hold check 1} &= \text{current launch edge} - \text{previous latch edge} \\ &= \quad 0\ \text{ns} - 2\ \text{ns} \\ &= -2\ \text{ns} \end{aligned}$$

$$\begin{aligned} \text{hold check 2} &= \text{next launch edge} - \text{current latch edge} \\ &= 5\ \text{ns} - 12\ \text{ns} \\ &= -7\ \text{ns} \end{aligned}$$

In this example, hold check two is too restrictive. The data is launched next by the edge at 10 ns and should check against the data captured by the current latch edge at 12 ns, which does not occur in hold check two. To correct the default analysis, you must use a start multicycle hold exception of one.

3.3.6 SDC (Clock and Exception) Assignments on Blackbox Ports

In the Quartus Prime Pro Edition software you can assign clock definitions and SDC exceptions on partition ports. Your design should follow the Hierarchical Design flow as a prerequisite.

Partition ports are inserted into your timing netlist as combinational nodes with names that persist in the netlist and cannot be optimized away. It is safe to refer to them as clock sources or `-through` points for SDC exceptions. You can also use them as `-from` and `-to` points in the `report_path` command.

If a port on `partition_a` is called `clk_divide` your SDC constraint would be `create_generated_clock –source clock -divide_by 2 top|partition_a|clk_divide`

If a set of ports on `partition_b` is called `data_input[0..7]` your SDC constraint would be `set_multicycle_path –from top|partition_a|data_reg* -through top|partition_b|data_input* 2`

Starting with Quartus Prime Pro Edition 16.0 you can use multiple `-through` clauses, which means you can, for example, specify paths that go through output ports of one partition and through the input ports of another, downstream partition.

To invoke this feature:

1. Compile your hierarchical design with appropriate partitions and ports defined or perform Analysis & Synthesis on your design.
2. Open the **RTL Viewer** and find the partition ports of interest.
3. Using the same names as in the **RTL Viewer**, add clock and SDC constraints to the SDC file for your project. You may also use wildcards to refer to more than one port in SDC exceptions.
4. Recompile the design. The Quartus Prime software will now honor your new definitions and constraints.

This feature is primarily designed for, but is not limited to, aiding the emulation of ASICs using FPGAs. In this type of design, clock networks often span multiple hierarchies of partitions. The enable circuitry or the clock-dividing circuitry is removed from the netlist, since it cannot be easily emulated on Intel FPGAs. For such clock networks, there needs to be the ability to define different versions of the clock signal in places where such circuitry was removed. You must design and place your partitions strategically, and then define the appropriate ports on these partitions. Best practice is for your ports and partitions to coincide with the part of the clock network which contains the special circuitry. Your emulated ASIC netlist can be edited manually (or by script) to annotate appropriate clock definitions and clock relationships. This feature is not limited to ASIC emulation; you can use it in any projects where arbitrary locations on paths need constrained timing or defined clock sources.

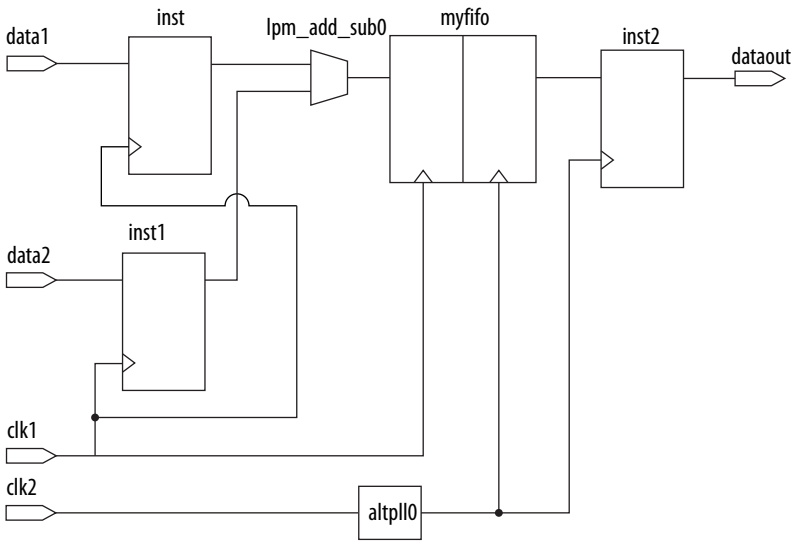
Related Links

- Input Constraints on page 65
  Input constraints allow you to specify all the external delays feeding into the device. Specify input requirements for all input ports in your design.
- Output Constraints on page 66
  Output constraints allow you to specify all external delays from the device for all output ports in your design.
- Planning for Hierachical and Team-Based Design

3.3.7 A Sample Design with SDC File

An example circuit that includes two clocks, a phase-locked loop (PLL), and other common synchronous design elements helps demonstrate how to constrain your design with an SDC file.

Figure 100. TimeQuest Constraint Example



The following SDC file contains basic constraints for the example circuit.

Example 6. Basic SDC Constraints

```
# Create clock constraints
create_clock -name clockone -period 10.000 [get_ports {clk1}]
create_clock -name clocktwo -period 10.000 [get_ports {clk2}]
# Create virtual clocks for input and output delay constraints
create clock -name clockone_ext -period 10.000
create clock -name clockone_ext -period 10.000
derive_pll_clocks
# derive clock uncertainty
derive_clock_uncertainty
# Specify that clockone and clocktwo are unrelated by assinging
# them to seperate asynchronus groups
set_clock_groups \
  -asynchronous \
  -group {clockone} \
  -group {clocktwo altpll0|altpll_component|auto_generated|pll1|clk[0]}
# set input and output delays
set_input_delay -clock { clockone_ext } -max 4 [get_ports {data1}]set_input_delay
-clock { clockone_ext } -min -1 [get_ports {data1}]
set_input_delay -clock { clockone_ext } -max 4 [get_ports {data2}]set_input_delay
-clock { clockone_ext } -min -1 [get_ports {data2}]
set_output_delay -clock { clocktwo_ext } -max 6 [get_ports {dataout}]
set_output_delay -clock { clocktwo_ext } -min -3 [get_ports {dataout}]
```

The SDC file contains the following basic constraints you should include for most designs:

- Definitions of `clockone` and `clocktwo` as base clocks, and assignment of those settings to nodes in the design.
- Definitions of `clockone_ext` and `clocktwo_ext` as virtual clocks, which represent clocks driving external devices interfacing with the FPGA.
- Automated derivation of generated clocks on PLL outputs.
- Derivation of clock uncertainty.
- Specification of two clock groups, the first containing `clockone` and its related clocks, the second containing `clocktwo` and its related clocks, and the third group containing the output of the PLL. This specification overrides the default analysis of all clocks in the design as related to each other.
- Specification of input and output delays for the design.

Related Links

Asynchronous Clock Groups on page 63
For more information about asynchronous clock groups.

3.4 Running the TimeQuest Analyzer

When you compile a design, the TimeQuest timing analyzer automatically performs multi-corner signoff timing analysis after the Fitter has finished.

- To open the TimeQuest analyzer GUI directly from the Quartus Prime software GUI, click **TimeQuest Timing Analyzer** on the Tools menu.
- To peform or repeat multi-corner timing analysis from the Quartus Prime GUI, click **Processing ➤ Start ➤ Start TimeQuest Timing Analyzer**.
- To perform multi-corner timing analysis from a system command prompt, type `quartus_sta <options><project_name>`.
- To run the TimeQuest analyzer as a stand-alone GUI application, type the following command at the command prompt:`quartus_staw`.
- To run the TimeQuest analyzer in interactive command-shell mode, type the following command at a system command prompt: `quartus_sta -s <options><project_name>`.

The following table lists the command-line options available for the `quartus_sta` executable.

Table 18. Summary of Command-Line Options

Command-Line Option	Description
`-h | --help`	Provides help information on quartus_sta.
`-t <script file> | --script=<script file>`	Sources the *<script file>*.
`-s | --shell`	Enters shell mode.
`--tcl_eval <tcl command>`	Evaluates the Tcl command *<tcl command>*.

continued...

Command-Line Option	Description

`--do_report_timing`	For all clocks in the design, run the following commands:
	`report_timing -npaths 1 -to_clock $clock`
	`report_timing -setup -npaths 1 -to_clock $clock`
	`report_timing -hold -npaths 1 -to_clock $clock`
	`report_timing -recovery -npaths 1 -to_clock $clock`
	`report_timing -removal -npaths 1 -to_clock $clock`

`--force_dat`	Forces an update of the project database with new delay information.
`--lower_priority`	Lowers the computing priority of the `quartus_sta` process.
`--post_map`	Uses the post-map database results.
`--sdc=<SDC file>`	Specifies the SDC file to use.
`--report_script=<script>`	Specifies a custom report script to call.
`--speed=<value>`	Specifies the device speed grade used for timing analysis.
`--tq2pt`	Generates temporary files to convert the TimeQuest analyzer SDC file(s) to a PrimeTime SDC file.
`-f <argument file>`	Specifies a file containing additional command-line arguments.
`-c <revision name> | --rev=<revision_name>`	Specifies which revision and its associated Quartus Prime Settings File (`.qsf`) to use.
`--multicorner`	Specifies that all slack summary reports be generated for both slow- and fast-corners.
`--multicorner[=on|off]`	Turns off multicorner timing analysis.
`--voltage=<value_in_mV>`	Specifies the device voltage, in mV used for timing analysis.
`--temperature= <value_in_C>`	Specifies the device temperature in degrees Celsius, used for timing analysis.
`--parallel [=<num_processors>]`	Specifies the number of computer processors to use on a multiprocessor system.
`--64bit`	Enables 64-bit version of the executable.
`--mode=implement|finalize`	Regulates whether TimeQuest performs a reduced-corner analysis for intermediate operations (`implement`),or a four-corner analysis for final Fitter optimization and placement (`finalize`).

Related Links

- Constraining and Analyzing with Tcl Commands on page 118
  For more information about using Tcl commands to constrain and analyze your design
- Recommended Flow for First Time Users on page 42
  For more information about steps to perform before opening the TimeQuest analyzer.

3.4.1 Quartus Prime Settings

Within the Quartus Prime software, there are a number of quick steps for setting up your design with TimeQuest. You can modify the appropriate settings in **Assignments ➤ Settings**.

In the **Settings** dialog box, select **TimeQuest Timing Analyzer** in the **Category** list.

The **TimeQuest Timing Analyzer** settings page is where you specify the title and location for a Synopsis Design Constraint (SDC) file. The SDC file is an inudstry standard format for specifying timing constraints. If no SDC file exists, you can create one based on the instructions in this document. The Quartus Prime software provides an SDC template you can use to create your own.

The following TimeQuest options should be on by default:

- Enable multicorner timing analysis—Directs the TimeQuest analyzer to analyze all the timing models of your FPGA against your constraints. This is required for final timing sign-off. Unchecked, only the slow timing model is be analyzed.
- Enable common clock path pessimism removal— Prevents timing analysis from over-calculating the effects of **On-Die Variation**. This makes timing better, and there really is no reason for this to be disabled.
- Report worst-case paths during compilation—This optional setting displays summary of the worst paths in your timing report. This type of path analysis is covered in more detail later in this document. While useful, this summary can increase the size of the <*project*>.sta.rpt with all of these paths.
- Tcl script file for custom reports—This optional setting should prove useful later, allowing you to add custom reports to create a custom analysis. For example, if you are only working on a portion of the full FPGA, you may want additional timing reports that cover that hierarchy.

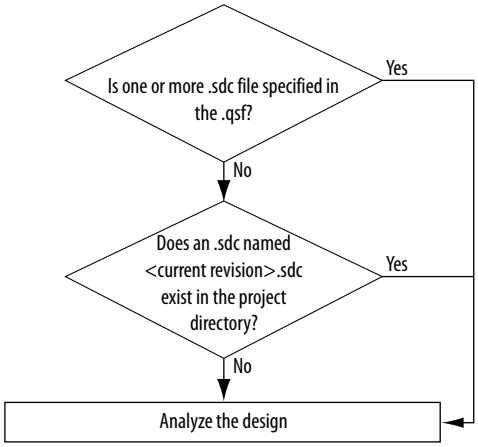
Note: In addition, certain values are set by default. The default duty-cycle is 50% and the default clock frequency is 1Ghz.

3.4.2 SDC File Precedence

The Fitter and the TimeQuest analyzer process SDC files in the order you specify in the Quartus Prime Settings File (.qsf). You can add and remove SDC files to process and specify the order they are processed from the **Assignments** menu.

Click **Settings**, then **TimeQuest Timing Analyzer** and add or remove SDC files, or specify a processing order in the **SDC files to include in the project** box. When you create a new SDC file for a project, you must add it to the project for it to be read during fitting and timing analysis. If you use the Quartus Prime Text Editor to create an SDC file, the option to add it to the project is enabled by default when you save the file. If you use any other editor to create an SDC file, you must remember to add it to the project. If no SDC files are listed in the .qsf, the Quartus Prime software looks for an SDC named <*current revision*>.sdc in the project directory. When you use IP from Intel, and some third-parties, the SDC files are often included in a project through an intermediate file called a Quartus Prime IP File (.qip). A .qip file points to all source files and constraints for a particular IP. If SDC files for IP blocks in your design are included through with a .qip, do not re-add them manually. An SDC file can also be added from a Quartus Prime IP File (.qip) included in the .qsf.

Figure 101. .sdc File Order of Precedence



Note: If you type the `read_sdc` command at the command line without any arguments, the TimeQuest analyzer reads constraints embedded in HDL files, then follows the SDC file precedence order.

The SDC file must contain only SDC commands that specify timing constraints. There are some techniques to control which constraints are applied during different parts of the compilation flow. Tcl commands to manipulate the timing netlist or control the compilation must be in a separate Tcl script.

3.5 Understanding Results

Knowing how your constraints are displayed when analyzing a path is one of the most important skills of timing analysis. This information completes your understanding of timing analysis and lets you correlate the SDC input to the back-end analysis, and determine how the delays in the FPGA affect timing.

3.5.1 Iterative Constraint Modification

Sometimes it is useful to change an SDC constraint and reanalyze the timing results. This flow is particularly common when you are creating timing constraints and want to ensure that they will be applied appropriately during compilation and timing analysis.

Use the following steps when you iteratvely modify constraints:

1. Open the TimeQuest Timing Analyzer
2. Generate the appropriate reports.
3. Analyze your results
4. Edit your SDC file and save
5. Double-click **Reset Design**
6. Generate the appropriate reports.
7. Analyze your results
8. Repeat steps 4-7 as necessary.

Open the TimeQuest Timing Analyzer—It is most common to use this interactive approach in the TimeQuest GUI. You can also use the command-line shell mode, but it does not include some of the time-saving automatic features in the GUI.

Generate the appropriate reports —Use the **Report All Summaries** task under **Macros** to generate setup, hold, recovery, and removal summaries, as well as minimum pulse width checks, and a list of all the defined clocks. These summaries cover all constrained paths in your design. Especially when you are modifying or correcting constraints, you should also perform the Diagnostic task to create reports to identify unconstrained parts of your design, or ignored constraints. Double-click any of the report tasks to automatically run the three tasks under **Netlist Setup** if they haven't already run. One of those tasks reads all SDC files.

Analyze your results—When you are modifying or correcting constraints, review the reports to find any unexpected results. For example, a cross-domain path might indicate that you forgot to cut a transfer by including a clock in a clock group.

Edit your SDC file and save it—Create or edit the appropriate constraints in your SDC files. If you edit your SDC file in the Quartus Prime Tex Editor, you can benefit from tooltips showing constraint options, and dialog boxes that guide you when creating constraints.

Reset the design—Double click **Reset Design** task to remove all constraints from your design. Removing all constraints from your design prepares it to reread the SDC files, including your changes.

Be aware that this method just performs timing analysis using new constraints, but the fit being analyzed has not changed. The place-and-route was performed with the old constraints, but you are analyzing with new constraints, so if something is failing timing against these new constraints,you may need to run place-and-route again.

For example, the Fitter may concentrate on a very long path in your design, trying to close timing. For example, you may realize that a path runs at a lower rate, and so have added `set_multicycle_path` assignments to relax the relationship (open the window when data is valid). When you perform TimeQuest analysis iteratively with these new multicycles, new paths replace the old. The new paths may have sub-optimal placement since the Fitter was concentrating on the previous paths when it ran, because they were more critical. The iterative method is recommend for getting your SDC files correct, but you should perform a full compilation to see what the Quartus Prime software can do with those constraints.

Related Links

Relaxing Setup with set_multicyle_path on page 74
A common type of multicycle exception occurs when the data transfer rate is slower than the clock cycle.

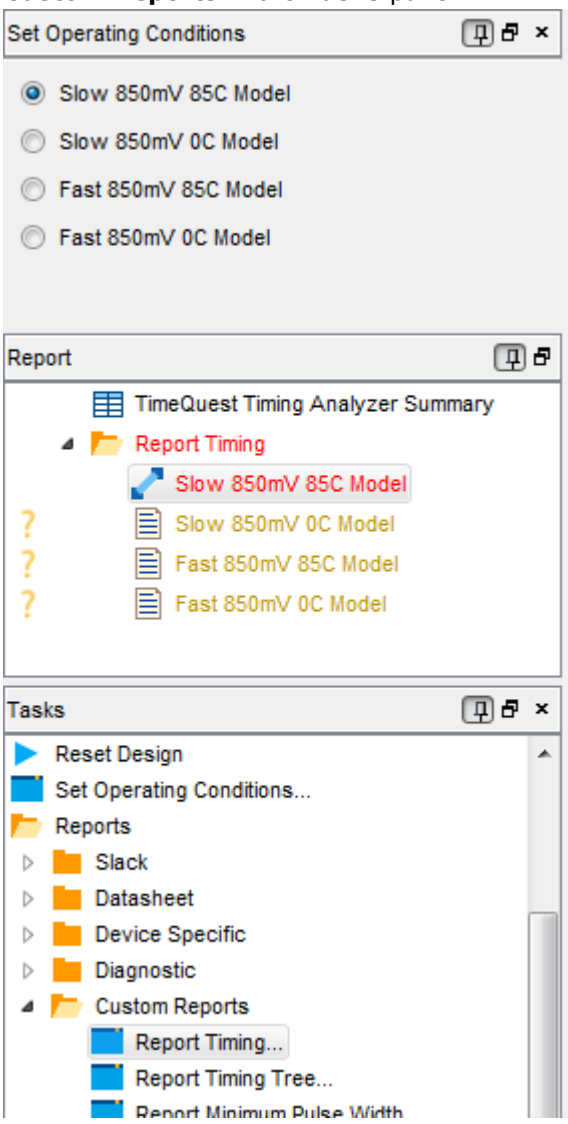
3.5.2 Set Operating Conditions Dialog Box

You can select different operating conditions to analyze from those used to create the existing timing netlist.

Operating conditions consist of voltage and temperature options that are used together. You can run timing analysis for different delay models without having to delete the existing timing netlist. The TimeQuest analyzer supports multi-corner

timing analysis which you can turn on in the dialog box of the command you are performing. A control has been added to the TimeQuest UI where you can select operating conditions and analyze timing for combinations of corners.

Select a voltage/temperature combination and double-click **Report Timing** under **Custom Reports** in the **Tasks** pane.



Reports that fail timing appear in red type, reports that pass appear in black type. Reports that have not yet been run are in gold with a question mark (?). Selecting another voltage/temperature combination creates a new report, but any reports previously run persist.

You can use the following context menu options to generate or regenerate reports in the **Report** window:

- **Regenerate**—Regenerate the selected report.
- **Generate in All Corners**—Generate a timing report using all corners.
- **Regenerate All Out of Date**—Regenerate all reports.
- **Delete All Out of Date**—Flush all the reports that have been run to clear the way for new reports with modifications to timing.

Each operating condition generates its own set of reports which appear in their own folders under the **Reports** list. Reports that have not yet been generated display a '**?**' icon in gold. As each report is generated, the folder is updated with the appropriate output.

Note: Reports for a corner not being generated persist until that particular operating condition is modifyed and a new report is created.

3.5.3 Report Timing (Dialog Box)

Once you are comfortable with the **Report All Summaries** command, the next tool in the TimeQuest analyzer toolbox is **Report Timing...**.

The TimeQuest analyzer displays reports in the **Report** pane, and is similar to a table of contents for all the reports created. Selecting any name in the **Report** panel displays that report in the main viewing pane. Below is a design with the **Summary (Setup)** report highlighted:

The main viewing pane shows the Slack for every clock domain. Positive slack is good, saying these paths meet timing by that much. The End Point TNS stands for Total Negative Slack, and is the sum of all slacks for each destination and can be used as a relative assessment of how much a domain is failing.

However, this is just a summary. To get details on any domain, you can right-click that row and select **Report Timing...**.

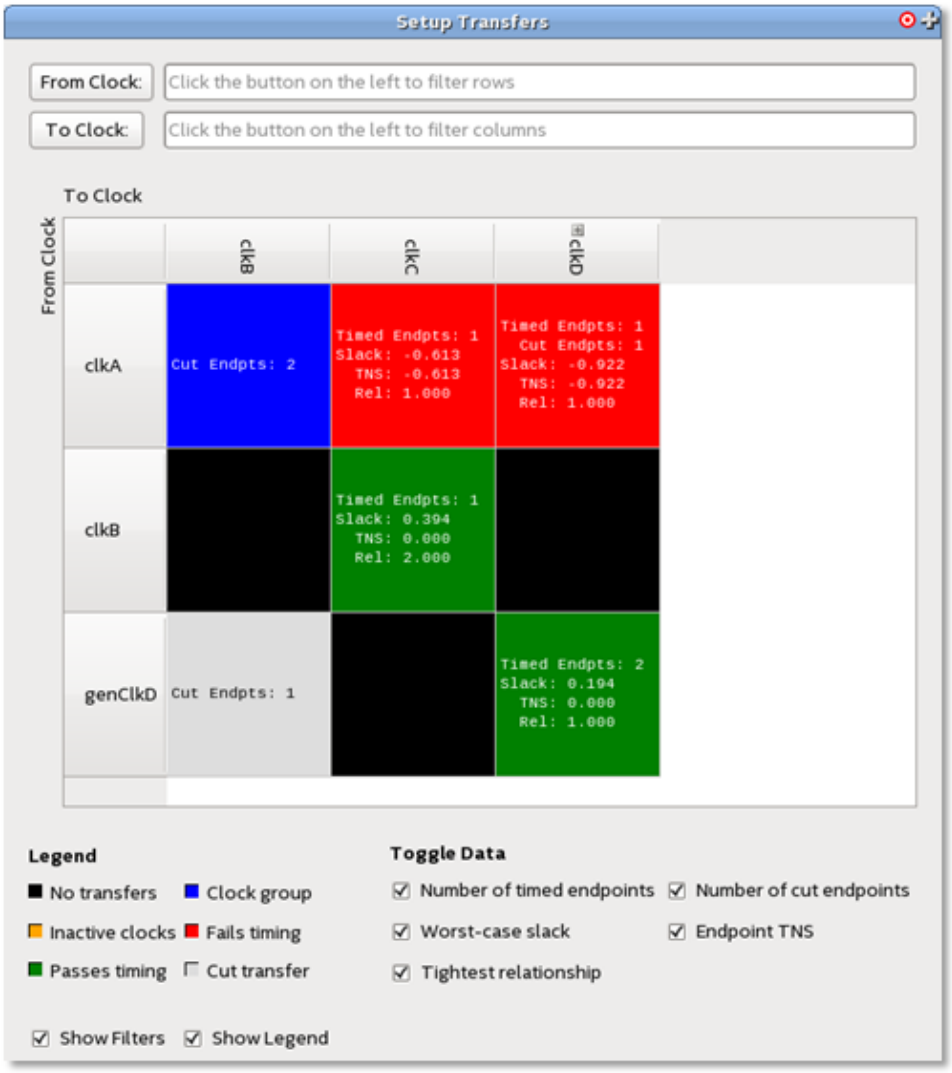
The **Report Timing** dialog box appears, auto-filled with the **Setup** radio button selected and the **To Clock** box filled with the selected clock. This occurs because you were viewing the **Setup Summary** report, and right-clicked on that particular clock. As such, the worst 10 paths where that is the destination clock were reported.You can modify the settingsin various ways, such as increasing the number of paths to report, adding a **Target** filter, adding a **From Clock**, writing the report to a text file, etc.

Note that any `report_timing` command can be copied from the **Console** at the bottom into a user-created Tcl file, so that you can analyze specific paths again in the future without having to negotiate the TimeQuest analyzer UI. This is often done as users become more comfortable with TimeQuest and find themselves analyzing the same problematic parts of their design over and over, but is not required. Many complex designs successfully use TimeQuest as a diving tool, i.e. just starting with summaries and diving down into the failing paths after each compile.

3.5.4 Report CDC Viewer Command

The Clock Domain Crossing (CDC) Viewer creates a graphical display which shows either setup, hold, recovery, or removal analysis of all clock transfers in your design. You open this report in the TimeQuest GUI by clicking **Reports ➤ Diagnostic ➤ Report CDC Viewer** in the TimeQuest Timing Analyzer. This creates a folder of **CDC Viewer** reports, containing one report for each analysis type.

Figure 102. Setup Transfers Report



Report Panels

The reports consists of:

- Filter boxes for filtering **From Clock:** and **To Clock:** values. Clicking on the **From Clock:** or **To Clock:** opens the **Name Finder** dialog box.
- A color coded grid which displays the clock transfer status. Status colors are defined in the **Legend**.
  - The clock headers list each clock with transfers in the design. If the name of the clock is too long, the display is truncated, but the full name can be seen in a tool tip or by resizing the clock header cell.
  - Generated clocks are represented as children of the clock they are derived from. A '+' icon next to a clock name indicates there are generated clocks associated. Clicking on that clock header will display the generated clocks associated with that clock.
- A **Legend** and **Toggle Data** section for controlling the grid display output.
- You can use the **Show Filters** and **Show Legend** controls to turn Filters and Legend on or off.

Transfer Cell Content

Each block in the grid is referred to as a transfer cell. Each transfer cell uses color and text to display important details of the paths involved in a transfer. The color coding represents the following states:

- Black—No transfers. There are no paths crossing between the source and destination clock of this cell.
- Green—Passes timing. All timing paths in this transfer, that have not been cut, meet their timing requirements.
- Red—Fails timing. One or more of the timing paths in the transfer do not meet their timing requirements. If the transfer is between unrelated clocks, the paths likely need to be synchronized by a synchronizer chain.
- Blue—Clock groups. The source and destination clocks of these transfers have been cut by means of asynchronous clock groups.
- Gray—Cut transfer. All paths in this transfer have been cut by false paths. The result of this is that these paths are not considered during timing analysis.
- Orange—Inactive clocks. One of the clocks involved in the transfer has been marked as an inactive clock (with the `set_active_clocks` command). Such transfers are ignored by the TimeQuest analyzer.

The text in each transfer cell contains data specific to each transfer. Types of data on display can be turned on or off with the **Toggle Data** boxes but you can mouse over any cell to see the full text. These data types are:

- **Number of timed endpoints** between clocks— The number of timed, endpoint-unique paths in the transfer. A path being "timed" means that it was analyzed during timing analysis. Only paths with unique endpoints count towards this total.
- **Number of cut endpoints** between clocks— The number of cut endpoint-unique paths, instead of timed ones. These paths have been cut by either a false path or clock group assignment. Such paths are skipped during timing analysis.
- **Worst-case slack** between clocks— The worst-case slack among all endpoint-unique paths in the transfer.
- **Total negative slack** between clocks— The sum of all negative slacks among all endpoint-unique paths in this transfer.
- **Tightest relationship** between clocks— The lowest-valued setup / hold / recovery / removal relationship between the two clocks in this transfer, depending on the analysis mode of the report

Transfer Cell Operations

Right-click menus allow you to perform operations on transfer cells and clock headers. When the operation is a TimeQuest report or SDC command, a dialog box opens prepopulated with the contents of the transfer cell.

Transfer cell operations include:

- **Copy**—Copies the contents of the transfer cell to the clipboard.
- **Copy (include children)**—Copies the name of the chose clock header, and the names of each of its derived clocks. This option only appears for clock headers with generated clocks.
- **Report Timing**—Not available for transfer cells with no valid paths (gray or black cells).
- **Report Endpoints**—Not available for transfer cells with no cut paths (gray or black cells).
- **Report False Path**—Not available for transfer cells with no valid paths (black cells).
- **Report Exceptions**
- **Report Exceptions (with clock groups)**—Only available for clock group transfers (blue cells)
- **Set False Path**
- **Set Multicycle Path**
- **Set Min Delay**
- **Set Max Delay**
- **Set Clock Uncertainty**

Clock header operations include:

- **Copy**—Copies the contents of the clock header to the clipboard.
- **Expand/Collapse All Rows/Columns**—Shows or hides all derived clocks in the grid.
- **Create Slack Histogram**—Generates a slack histogram report for the selected clock.
- **Report Timing From/To Clock**—Generates a timing report for the selected clock. If the clock has not been expanded to display its derived clocks, all clocks derived from the selected clock are included in the timing report as well. To prevent this, expand the clock before right-clicking it.
- **Remove Clock(s)**—Removes the selected clock from the design. If the clock has not been expanded, all clocks derived from the selected clock are also removed.

As with other TimeQuest reports, you may view CDC Viewer output in four formats:

- A report panel in the TimeQuest analyzer
- Output in the TimeQuest Tcl console
- A plain-text file
- An HTML file that can be viewed in a web browser.

The TimeQuest analyzer report panel is the recommended format.

3.5.4.1 Report Custom CDC Viewer Command

Allows you to configure and display a customized report that creates a customized clock domain crossing report to either a file, the Tcl console, or a report panel. This report displays the results of setup, hold, recovery, and removal checks on clock domain crossing transfers. You open this report in the TimeQuest analyzer by clicking **Reports ➤ Custom Reports ➤ Report Custom CDC Viewer**.

Note:

You can click the **Pushpin** button  to keep the **Report False Path**, **Report Timing**, and **Report Endpoints** dialog boxes open after you generate a report. You can use this feature to fine tune your report settings or quickly create additional reports. You can click **Close** to close the dialog box at any time.

Filter boxes for filtering **From Clock:** and **To Clock:** values. Clicking on the buttons next to each box opens the **Name Finder** dialog box.

Analysis Type

The CDC Viewer can analyze any combination of **Setup**, **Hold**, **Recovery**, or **Removal**.

Scripting Information
Keyword: `report_cdc_viewer` **Settings:** `-setup|-hold|-recovery|-removal`

Transfer Filtering

By default, all transfer types are included in CDC Viewer reports:

- **Timed transfers**—Passing or failing
- **Fully cut transfers**— Transfers where all paths are false paths.
- **Clock groups**
- **Inactive clocks**

You can use these options to narrow down which kinds of transfers to show, or by adding the desired transfer types as options: –timed, –fully_cut, –clock_groups, and –inactive. If none are specified, all transfer types are shown.

Scripting Information
Keyword:`report_cdc_viewer` **Settings:**`-timed|-fully_cut|-clock_groups|-inactive`

By default, only clocks are shown are clocks that launch or latch paths that are launched or latched by clocks other than themselves. Turning on **Non-crossing transfers** shows clocks with transfers to or from themselves.

Scripting Information
Keyword:`report_cdc_viewer` **Settings:**`-show_non_crossing`

Note: In grid-formatted reports, clocks with non-crossing transfers are always shown as long as they have transfers between other clocks too.

If you specify a value in the **Maximum slack limit** box, only paths with slack less than the value are displayed. If this option is not included, paths of any slack value will be included in the report.

Scripting Information
Keyword:`report_cdc_viewer` **Settings:**`-less_than_slack`

Grid Options

In grid-formatted reports, the grid can be configured to display clocks as either a flat list or in a hierarchy where generated clocks are displayed as children of the clock they are derived from. Turn on **Fold clocks on hierarchy** to enable this nested display.

Scripting Information
Keyword:`report_cdc_viewer` **Settings:**`-hierarchy`

By default, clocks that launch or latch to no paths are not shown in grid-based reports. You can show these clocks by turning on the **Show empty transfers** option.

Scripting Information
Keyword:`report_cdc_viewer` **Settings:**`-show_empty`

Output

Allows you to specify where you want to save the report and how much detail you want in the report. You can select one or more of the following settings:

- **Report panel name**— Directs the TimeQuest analyzer to generate a report panel with the specified name. The default report name is Report Timing.

Scripting Information
Keyword: `report_cdc_viewer` **Settings:** `-panel_name<reportname>`

- **Enable multi-corner reports**— Allows you to enable or disable multi-corner timing analysis. This option is on by default.

Scripting Information
Keyword: `report_cdc_viewer` **Settings:** `-multi_corner`

- **File name**— Directs the TimeQuest analyzer to save the report to your local disk as a text file with the specified file name. To save a report in HTML, end the filename with ".html".

Scripting Information
Keyword: `report_cdc_viewer` **Settings:** `-file<filename>`

- **Format**— Specifies that the generated report file is formatted as a list of clock transfers rather than the default grid panel.

Scripting Information
Keyword: `report_cdc_viewer` **Settings:** `-list`

- Under **File options** you can specify whether the TimeQuest analyzer overwrites an existing file (the default setting) or appends the content to an existing file.

Scripting Information
Keyword: `report_cdc_viewer` **Settings:** `-append|-overwrite`

- **Console**— Specifies whether the report appears as information messages in the **Console**.

Scripting Information
Keyword: `report_cdc_viewer` **Settings:** `-stdout`

3.5.5 Analyzing Results with Report Timing

Report Timing is one of the most useful analysis tools in TimeQuest. Many designs require nothing but this command. In the TimeQuest analyzer, this command can be accessed from the **Tasks** menu , from the **Reports ➤ Custom Reports** menu, or by right-clicking on nodes or assignments in TimeQuest.

You can review all of the options for **Report Timing** by typing `report_timing -long_help` in the TimeQuest console.

Clocks

The **From Clock** and **To Clock** in the **Clocks** box are used to filter paths where the selected clock is used as the launch or latch. The pull-down menu allows you to choose from existing clocks (although admittedly has a "limited view" for long clock names).

Targets

The boxes in the **Targets** box are targets for the **From Clock** and **To Clock**settings, and allow you to report paths with only particular endpoints. These are usually filled with register names or I/O ports, and can be wildcarded. For example, you might use the following to only report paths within a hierarchy of interest:

```
report_timing -from *|egress:egress_inst|* -to *|
egress:egress_inst|* -(other options)
```

If the **From**, **To**, or **Through** boxes are empty, then the TimeQuest analyzer assumes you are refering to all possible targets in the device, which can also be represented with a wildcard (*). The **From** and **To** options cover the majority of situations. The**Through** option is used to limit the report for paths that pass through combinatorial logic, or a particular pin on a cell. This is seldom used, and may not be very reliable due to combinatorial node name changes during synthesis. Clicking the browse **Browse** box after each target opens the **Name Finder** dialog box to search for specific names. This is especially useful to make sure the name being entered matches nodes in the design, since the **Name Finder** can immediately show what matches a user's wildcard.

Analysis type

The **Analysis type** options are **Setup**, **Hold**, **Recovery**, or **Removal**. These will be explained in more detail later, as understanding them is the underpinning of timing analysis.

Output

The **Detail** level, is an option often glanced over that should be understood. It has four options, but I will only discuss three.

The first level is called **Summary**, and produces a report which only displays Summary information such as

- **Slack**
- **From Node**
- **To Node**
- **Launch Clock**
- **Latch Clock**
- **Relationship**
- **Clock Skew**
- **Data Delay**

The **Summary** report is always reported with more detailed reports, so the user would choose this if they want less info. A good use for summary detail is when writing the report to a text file, where **Summary** can be quite brief.

The next level is **Path only**. This report displays all the detailed information, except the **Data Path** tab displays the clock tree as one line item. This is useful when you know the clock tree is correct, details are not relevant. This is common for most paths within the FPGA. A useful data point is to look at the **Clock Skew** column in the **Summary** report, and if it's a small number, say less than +/-150ps, then the clock tree is well balanced between source and destination.

If there is clock skew, you should select the **Full path** option.. This breaks the clock tree out into explicit detail, showing every cell it goes through, including such things as the input buffer, PLL, global buffer (called CLKCTRL_), and any logic. If there is clock skew, this is where you can determine what is causing the clock skew in your design. The **Full path** option is also recommended for I/O analysis, since only the source clock or destination clock is inside the FPGA, and therefore its delay plays a critical role in meeting timing.

The Data Path tab of a detailed report gives the delay break-downs, but there is also useful information in the **Path Summary** and **Statistics** tabs, while the **Waveform** tab is useful to help visualize the **Data Path** analysis. I would suggest taking a few minutes to look at these in the user's design. The whole analysis takes some time to get comfortable with, but hopefully is clear in what it's doing.

Enable multi corner reports allows you to enable or disable multi-corner timing analysis. This option is on by default.

Report Timing also has the **Report panel name**, which displays the name used in TimeQuest's **Report** section. There is also an optional **File name** switch, which allows you to write the information to a file. If you append .htm as a suffix, the TimeQuest analyzer produces the report as HTML. The **File options** radio buttons allow you to choose between **Overwrite** and **Append** when saving the file.

Paths

The default value for **Report number of paths** is 10. Two endpoints may have a lot of combinatorial logic between them and might have many different paths. Likewise, a single destination may have hundreds of paths leading to it. Because of this, you might list hundreds of paths, many of which have the same destination and might have the same source. By turning on **Pairs only** you can list only one path for each pair of source and destination. An even more powerful way to filter the report is limit the Maximum number of paths per endpoints. You can also filter paths by entering a value in the **Maximum slack limit** field.

Tcl command

Finally, at the bottom is the **Tcl command**field, which displays the Tcl syntax of what is run in TimeQuest. You can edit this directly before running the **Report Timing** command.

Note:

A useful addition is to addis the `-false_path` option to the command line string. With this option, only false paths are listed. A false path is any path where the launch and latch clock have been defined, but the path was cut with either a

set_false_path assignment or *set_clock_groups_assignment*. Paths where the launch or latch clock was never constrained are not considered false paths. This option is useful to see if a false path assignment worked and what paths it covers, or to look for paths between clock domains that should not exist. The **Task** window's **Report False Path** custom report is nothing more than **Report Timing** with the `-false_path` flag enabled.

3.5.6 Correlating Constraints to the Timing Report

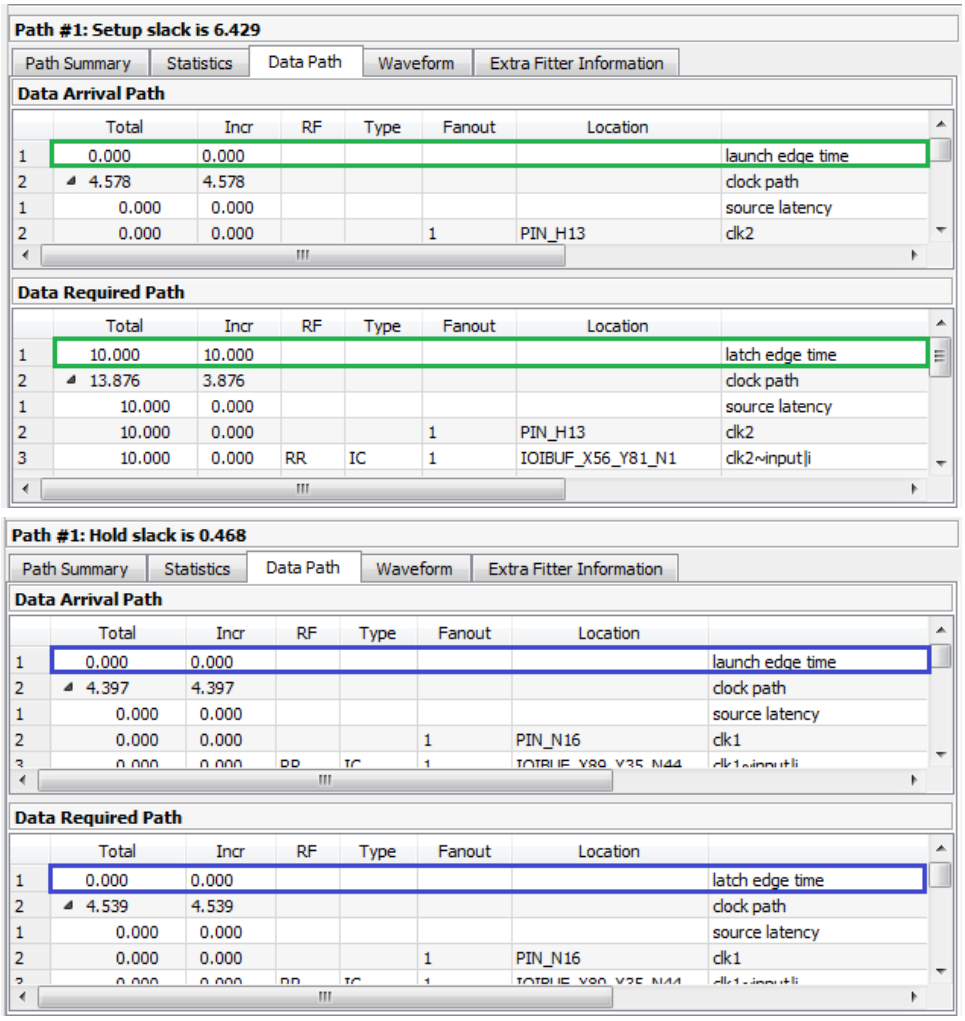
A critical part of timing analysis is how timing constraints appear in the **Report Timing** analysis. Most constraints only affect the launch and latch edges. Specifically, `create_clock` and `create_generated_clock` create clocks with default relationships. The command `set_multicycle_path` modifies those default relationships, while `set_max_delay` and `set_min_delay` are low-level overrides that explicitly tell TimeQuest what the launch and latch edges should be.

The following figures are from an example of the output of **Report Timing** on a particular path.

Initially, the design features a clock driving the source and destination registers with a period of 10ns. This results in a setup relationship of 10ns (launch edge = 0ns, latch edge = 10ns) and hold relationship of 0ns (launch edge = 0ns, latch edge = 0ns) from the command:

```
create_clock -name clocktwo -period 10.000 [get_ports {clk2}]
```

Figure 103. Setup Relationship 10ns, Hold Relationship 0ns



Path #1: Setup slack is 6.429

Path Summary | Statistics | Data Path | Waveform | Extra Fitter Information

Data Arrival Path

	Total	Incr	RF	Type	Fanout	Location	
1	0.000	0.000					launch edge time
2	4.578	4.578					clock path
1	0.000	0.000					source latency
2	0.000	0.000			1	PIN_H13	clk2

Data Required Path

	Total	Incr	RF	Type	Fanout	Location	
1	10.000	10.000					latch edge time
2	13.876	3.876					clock path
1	10.000	0.000					source latency
2	10.000	0.000			1	PIN_H13	clk2
3	10.000	0.000	RR	IC	1	IOIBUF_X56_Y81_N1	clk2~input|i

Path #1: Hold slack is 0.468

Path Summary | Statistics | Data Path | Waveform | Extra Fitter Information

Data Arrival Path

	Total	Incr	RF	Type	Fanout	Location	
1	0.000	0.000					launch edge time
2	4.397	4.397					clock path
1	0.000	0.000					source latency
2	0.000	0.000			1	PIN_N16	clk1

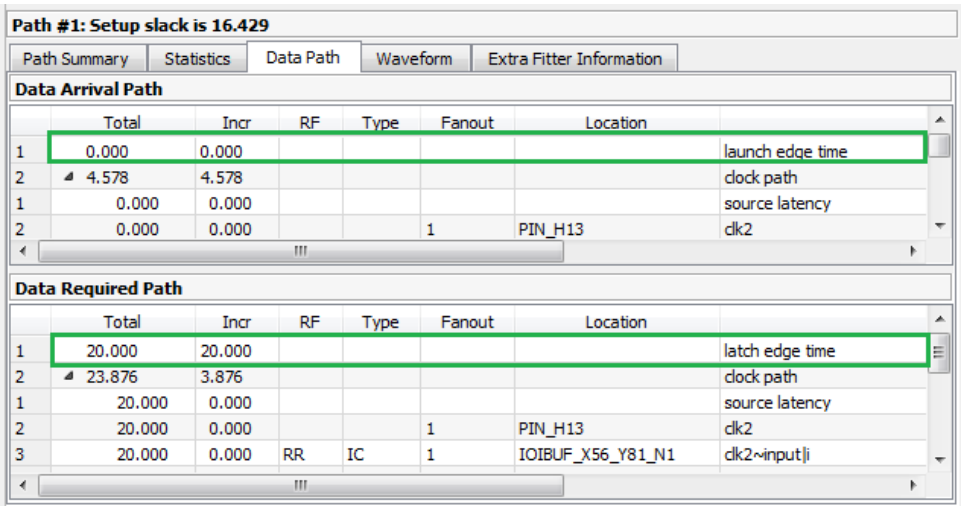
Data Required Path

	Total	Incr	RF	Type	Fanout	Location	
1	0.000	0.000					latch edge time
2	4.539	4.539					clock path
1	0.000	0.000					source latency
2	0.000	0.000			1	PIN_N16	clk1

In the next figure, using `set_multicycle_path` adds multicycles to relax the setup relationship, or open the window, making the setup relationship 20ns while the hold relationship is still 0ns:

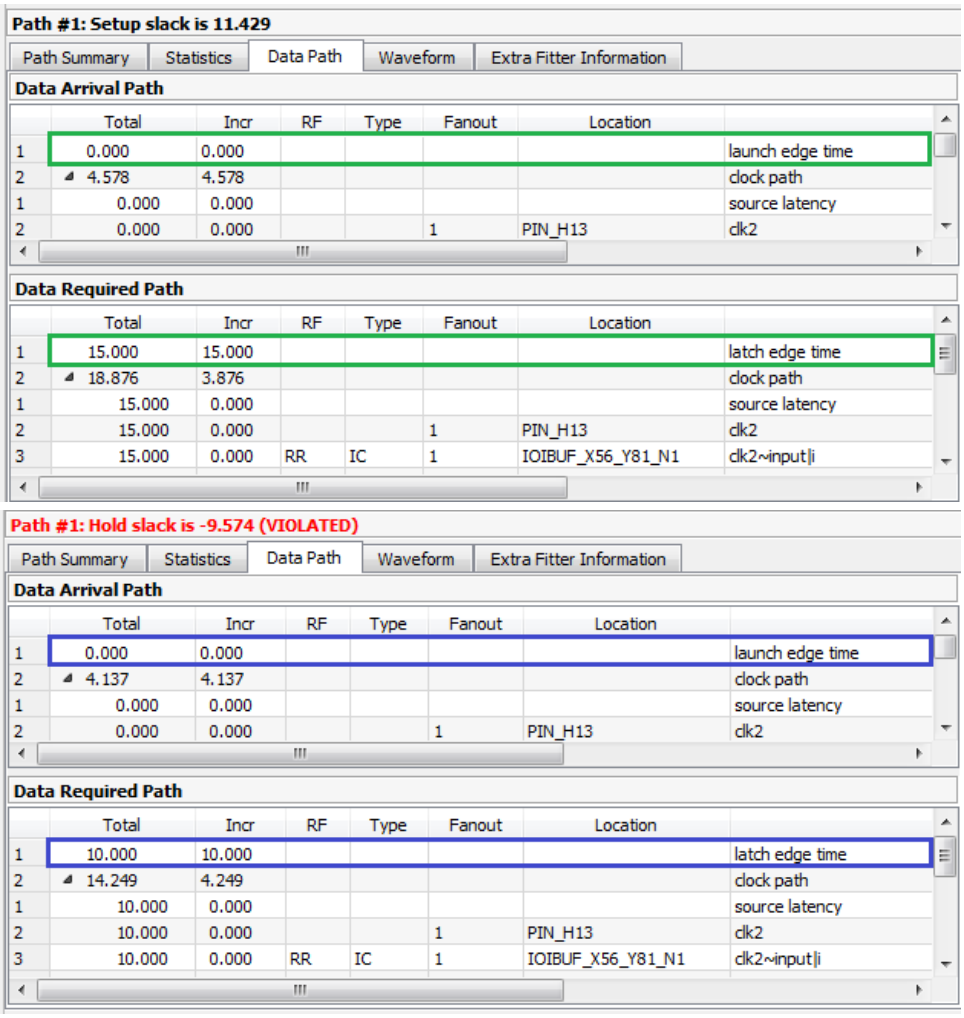
```
set_multicycle_path -from clocktwo -to clocktwo -setup -end 2
set_multicycle_path -from clocktwo -to clocktwo -hold -end 1
```

Figure 104. Setup Relationship 20ns



In the last figure, using the `set_max_delay` and `set_min_delay` constraints lets you explicitly override the relationships. Note that the only thing changing for these different constraints is the Launch Edge Time and Latch Edge Times for setup and hold analysis. Every other line item comes from delays inside the FPGA and are static for a given fit. Whenever analyzing how your constraints affect the timing requirements, this is the place to look.

Figure 105. Using set_max_delay and set_min_delay



For I/O, this all holds true except we must add in the `-max` and `-min` values. They are displayed as **iExt** or **oExt** in the **Type** column. An example would be an output port with a `set_output_delay -max 1.0` and `set_output_delay -min -0.5`:

Once again, the launch and latch edge times are determined by the clock relationships, multicycles and possibly `set_max_delay` or `set_min_delay` constraints. The value of `set_output_delay` is also added in as an **oExt** value. For outputs this value is part of the **Data Required Path**, since this is the external part of the analysis. The setup report on the left will subtract the `-max` value, making the setup relationship harder to meet, since we want the **Data Arrival Path** to be shorter than the **Data Required Path**. The `-min` value is also subtracted, which is why a negative number makes hold timing more restrictive, since we want the **Data Arrival Path** to be longer than the **Data Required Path**.

Related Links

Relaxing Setup with set_multicyle_path on page 74

A common type of multicycle exception occurs when the data transfer rate is slower than the clock cycle.

3.6 Constraining and Analyzing with Tcl Commands

You can use Tcl commands from the Quartus Prime software Tcl Application Programming Interface (API) to constrain, analyze, and collect information for your design. This section focuses on executing timing analysis tasks with Tcl commands; however, you can perform many of the same functions in the TimeQuest analyzer GUI. SDC commands are Tcl commands for constraining a design. SDC extension commands provide additional constraint methods and are specific to the TimeQuest analyzer. Additional TimeQuest analyzer commands are available for controlling timing analysis and reporting. These commands are contained in the following Tcl packages available in the Quartus Prime software:

- `::quartus::sta`
- `::quartus::sdc`
- `::quartus::sdc_ext`

Related Links

- ::quartus::sta
  For more information about TimeQuest analyzer Tcl commands and a complete list of commands, refer to Quartus Prime Help.
- ::quartus::sdc
  For more information about standard SDC commands and a complete list of commands, refer to Quartus Prime Help.
- ::quartus::sdc_ext
  For more information about Intel FPGA extensions of SDC commands and a complete list of commands, refer to Quartus Prime Help.

3.6.1 Collection Commands

The TimeQuest analyzer Tcl commands often return data in an object called a collection. In your Tcl scripts you can iterate over the values in collections to access data contained in them. The software returns collections instead of Tcl lists because collections are more efficient than lists for large sets of data.

The TimeQuest analyzer supports collection commands that provide easy access to ports, pins, cells, or nodes in the design. Use collection commands with any constraints or Tcl commands specified in the TimeQuest analyzer.

Table 19. SDC Collection Commands

Command	Description of the collection returned
`all_clocks`	All clocks in the design.
`all_inputs`	All input ports in the design.
`all_outputs`	All output ports in the design.
`all_registers`	All registers in the design.
`get_cells`	Cells in the design. All cell names in the collection match the specified pattern. Wildcards can be used to select multiple cells at the same time.

continued...

Command	Description of the collection returned
get_clocks	Lists clocks in the design. When used as an argument to another command, such as the -from or -to of set_multicycle_path, each node in the clock represents all nodes clocked by the clocks in the collection. The default uses the specific node (even if it is a clock) as the target of a command.
get_nets	Nets in the design. All net names in the collection match the specified pattern. You can use wildcards to select multiple nets at the same time.
get_pins	Pins in the design. All pin names in the collection match the specified pattern. You can use wildcards to select multiple pins at the same time.
get_ports	Ports (design inputs and outputs) in the design.

You can also examine collections and experiment with collections using wildcards in the TimeQuest analyzer by clicking **Name Finder** from the **View** menu.

3.6.1.1 Wildcard Characters

To apply constraints to many nodes in a design, use the "*" and "?" wildcard characters. The "*" wildcard character matches any string; the "?" wildcard character matches any single character.

If you make an assignment to node reg*, the TimeQuest analyzer searches for and applies the assignment to all design nodes that match the prefix reg with any number of following characters, such as reg, reg1, reg[2], regbank, and reg12bank.

If you make an assignment to a node specified as reg?, the TimeQuest analyzer searches and applies the assignment to all design nodes that match the prefix reg and any single character following; for example, reg1, rega, and reg4.

3.6.1.2 Adding and Removing Collection Items

Wildcards used with collection commands define collection items identified by the command. For example, if a design contains registers named src0, src1, src2, and dst0, the collection command [get_registers src*] identifies registers src0, src1, and src2, but not register dst0. To identify register dst0, you must use an additional command, [get_registers dst*]. To include dst0, you could also specify a collection command [get_registers {src* dst*}].

To modify collections, use the add_to_collection and remove_from_collection commands. The add_to_collection command allows you to add additional items to an existing collection.

add_to_collection Command

add_to_collection *<first collection> <second collection>*

Note: The add_to_collection command creates a new collection that is the union of the two specified collections.

The remove_from_collection command allows you to remove items from an existing collection.

remove_from_collection Command

remove_from_collection *<first collection> <second collection>*

You can use the following code as an example for using `add_to_collection` for adding items to a collection.

Adding Items to a Collection

```
#Setting up initial collection of registers
set regs1 [get_registers a*]
#Setting up initial collection of keepers
set kprs1 [get_keepers b*]
#Creating a new set of registers of $regs1 and $kprs1
set regs_union [add_to_collection $kprs1 $regs1]
#OR
#Creating a new set of registers of $regs1 and b*
#Note that the new collection appends only registers with name b*
# not all keepers
set regs_union [add_to_collection $regs1 b*]
```

In the Quartus Prime software, keepers are I/O ports or registers. A SDC file that includes `get_keepers` can only be processed as part of the TimeQuest analyzer flow and is not compatible with third-party timing analysis flows.

Related Links

- add_to_collection
- remove_from_collection
  For more information about the`add_to_collection` and `remove_from_collection` commands, refer to Quartus Prime Help.

3.6.1.3 Getting Other Information about Collections

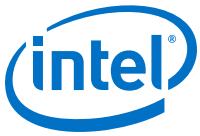
You can display the contents of a collection with the `query_collection` command. Use the `-report_format` option to return the contents in a format of one element per line. The `-list_format` option returns the contents in a Tcl list.

```
query_collection -report_format -all $regs_union
```

Use the `get_collection_size` command to return the size of a collection; the number of items it contains. If your collection is in a variable named `col`, it is more efficient to use `set num_items [get_collection_size $col]` than `set num_items [llength [query_collection -list_format $col]]`

3.6.1.4 Using the get_pins Command

The `get_pins` command supports options that control the matching behavior of the wildcard character (*). Depending on the combination of options you use, you can make the wildcard character (*) respect or ignore individual levels of hierarchy, which are indicated by the pipe character (|). By default, the wildcard character (*) matches only a single level of hierarchy.



These examples filter the following node and pin names to illustrate function:

- foo (a hierarchy level named foo)
- foo|dataa (an input pin in the instance foo)
- foo|datab (an input pin in the instance foo)
- foo|bar (a combinational node named bar in the foo instance)
- foo|bar|datac (an input pin to the combinational node named bar)
- foo|bar|datad (an input pin to the combinational node bar)

Table 20. Sample Search Strings and Search Results

Search String	Search Result
`get_pins *|dataa`	`foo|dataa`
`get_pins *|datac`	*<empty>*[3]
`get_pins *|*|datac`	`foo|bar|datac`
`get_pins foo*|*`	`foo|dataa, foo|datab`
`get_pins -hierarchical *|*|datac`	*<empty>*[3]
`get_pins -hierarchical foo|*`	`foo|dataa, foo|datab`
`get_pins -hierarchical *|datac`	`foo|bar|datac`
`get_pins -hierarchical foo|*|datac`	*<empty>*[3]
`get_pins -compatibility_mode *|datac`	`foo|bar|datac` [4]
`get_pins -compatibility_mode *|*|datac`	`foo|bar|datac`

The default method separates hierarchy levels of instances from nodes and pins with the pipe character (|). A match occurs when the levels of hierarchy match, and the string values including wildcards match the instance and/or pin names. For example, the command `get_pins <instance_name>|*|datac` returns all the `datac` pins for registers in a given instance. However, the command `get_pins *|datac` returns and empty collection because the levels of hierarchy do not match.

Use the `-hierarchical` matching scheme to return a collection of cells or pins in all hierarchies of your design.

For example, the command `get_pins -hierarchical *|datac` returns all the `datac` pins for all registers in your design. However, the command `get_pins -hierarchical *|*|datac` returns an empty collection because more than one pipe character (|) is not supported.

3 The search result is <*empty*> because the wildcard character (*) does not match more than one hierarchy level, indicated by a pipe character (|), by default. This command would match any pin named `datac` in instances at the top level of the design.

4 When you use `-compatibility_mode`, pipe characters (|) are not treated as special characters when used with wildcards.

The `-compatibility_mode` option returns collections matching wildcard strings through any number of hierarchy levels. For example, an asterisk can match a pipe character when using `-compatibility_mode`.

3.6.2 Using Fitter Overconstraints

Fitter overconstraints are adjusted timing assignments, typically used to try to overcome modelling inaccuracies, mis-correlation, or other deficiencies in compiler optimization. You may try to overconstrain setup and/or hold paths in the Fitter, to try to force more aggressive timing optimization of specific paths.

One way to assign Fitter overconstraints checks the name of the current executable, (either `quartus_fit` or `quartus_sta`) to apply different constraints for Fitter optimization and sign-off timing analysis.

```
set fit_flow 0
if { $::TimeQuestInfo(nameofexecutable) == "quartus_fit" } {
   set fit_flow 1
}
if {$fit_flow} {
  # Example Fitter overconstraint targeting specific nodes (restricts retiming)
  set_max_delay -from ${my_src_regs} -to ${my_dst_regs} 1ns
}
```

3.6.3 Locating Timing Paths in Other Tools

You can locate paths and elements from the TimeQuest analyzer to other tools in the Quartus Prime software.

Use the **Locate** or `Locate Path` command in the TimeQuest analyzer GUI or the `locate` command in the Tcl console in the TimeQuest analyzer GUI. Right-click most paths or node names in the TimeQuest analyzer GUI to access the **Locate** or **Locate Path** options.

The following commands are examples of how to locate the ten paths with the worst timing slack from TimeQuest analyzer to the **Technology Map Veiwer** and locate all ports matching data* in the **Chip Planner**.

Example 7. Locating from the TimeQuest Analyzer

```
# Locate in the Technology Map Viewer the ten paths with the worst slack
locate [get_timing_paths -npaths 10] -tmv
# locate all ports that begin with data in the Chip Planner
locate [get_ports data*] -chip
```

Related Links

locate
For more information on this command, refer to Quartus Prime Help.

3.7 Generating Timing Reports

The TimeQuest analyzer provides real-time static timing analysis result reports. The TimeQuest analyzer does not automatically generate most reports; you must create each report individually in the TimeQuest analyzer GUI or with command-line commands. You can customize in which report to display specific timing information, excluding fields that are not required.

Some of the different command-line commands you can use to generate reports in the TimeQuest analyzer and the equivalent reports shown in the TimeQuest analyzer GUI.

Table 21. TimeQuest Analyzer Reports

Command-Line Command	Report
`report_timing`	Timing report
`report_exceptions`	Exceptions report
`report_clock_transfers`	Clock Transfers report
`report_min_pulse_width`	Minimum Pulse Width report
`report_ucp`	Unconstrained Paths report

During compilation, the Quartus Prime software generates timing reports on different timing areas in the design. You can configure various options for the TimeQuest analyzer reports generated during compilation.

You can also use the `TIMEQUEST_REPORT_WORST_CASE_TIMING_PATHS` assignment to generate a report of the worst-case timing paths for each clock domain. This report contains worst-case timing data for setup, hold, recovery, removal, and minimum pulse width checks.

Use the `TIMEQUEST_REPORT_NUM_WORST_CASE_TIMING_PATHS` assignment to specify the number of paths to report for each clock domain.

An example of how to use the `TIMEQUEST_REPORT_WORST_CASE_TIMING_PATHS` and `TIMEQUEST_REPORT_NUM_WORST_CASE_TIMING_PATHS` assignments in the **`.qsf`** to generate reports.

Generating Worst-Case Timing Reports

```
#Enable Worst-Case Timing Report
set_global_assignment -name TIMEQUEST_REPORT_WORST_CASE_TIMING_PATHS ON
#Report 10 paths per clock domain
set_global_assignment -name TIMEQUEST_REPORT_NUM_WORST_CASE_TIMING_PATHS 10
```

Fmax Summary Report panel

Fmax Summary Report panel lists the maximum frequency of each clock in your design. In some designs you may see a note indicating "Limit due to hold check. Typically, Fmax is not limited by hold checks, because they are often same-edge relationships, and therefore independent of clock frequency, for example, launch = 0, latch = 0. However, if you have an inverted clock transfer, or a multicycle transfer such as setup=2, hold=0, then the hold relationship is no longer a same-edge transfer and changes as the clock frequency changes. The value in the **Restricted Fmax** column incorporates limits due to hold time checks in the situations described

previously, as well as minimum period and pulse width checks. If hold checks limit the Fmax more than setup checks, that is indicated in the **Note:** column as "Limit due to hold check".

Related Links

- ::quartus::sta
  For more information on this command, refer to Quartus Prime Help.
- TimeQuest Timing Analyzer Page
  For more information about the options you can set to customize TimeQuest analyzer reports.
- Timing Closure and Optimization
  For more information about timing closure recommendations.

3.8 Document Revision History

Table 22. Document Revision History

Date	Version	Changes
2017.05.08	17.0.0	• Added Using Fitter Overconstraints topic. • Added Clock Domain Crossing report topics
2016.10.31	16.1.0	• Implemented Intel rebranding. • Added support for `-blackbox` option with `set_input_delay`, `set_output_delay`, `remove_input_delay`, `remove_output_delay`.
2016.05.03	16.0.0	Added new topic: SCDS (Clock and Exception) Assignments on Blackbox Ports
2015.11.02	15.1.0	• Changed instances of *Quartus II* to *Quartus Prime*. • Added a description of running three- and four-corner analysis with `--mode=implement|finalize`. • Added description for new `set_operating_conditions` UI.
2015.05.04	15.0.0	Added and updated contents in support of new timing algorithms for Arria 10: • Enhanced Timing Analysis for Arria 10 • Maximum Skew (`set_max_skew` command) • Net Delay (`set_net_delay` command) • Create Generated Clocks (clock-as-data example)
2014.12.15	14.1.0	Major reorganization. Revised and added content to the following topic areas: • Timing Constraints • Create Clocks and Clock Constraints • Creating Generated Clocks • Creating Clock Groups • Clock Uncertainty • Running the TimeQuest Analyzer • Generating Timing Reports • Understanding Results • Constraining and Analyzing with Tcl Commands
August 2014	14.0a10.0	Added command line compliation requirements for Arria 10 devices.
June 2014	14.0.0	• Minor updates. • Updated format.
November 2013	13.1.0	• Removed HardCopy device information.

continued...

Date	Version	Changes
June 2012	12.0.0	• Reorganized chapter. • Added "Creating a Constraint File from Quartus Prime Templates with the Quartus Prime Text Editor" section on creating an SDC constraints file with the **Insert Template** dialog box. • Added "Identifying the Quartus Prime Software Executable from the SDC File" section. • Revised multicycle exceptions section.
November 2011	11.1.0	• Consolidated content from the Best Practices for the Quartus Prime TimeQuest Timing Analyzer chapter. • Changed to new document template.
May 2011	11.0.0	• Updated to improve flow. Minor editorial updates.
December 2010	10.1.0	• Changed to new document template. • Revised and reorganized entire chapter. • Linked to Quartus Prime Help.
July 2010	10.0.0	Updated to link to content on SDC commands and the TimeQuest analyzer GUI in Quartus Prime Help.
November 2009	9.1.0	Updated for the Quartus Prime software version 9.1, including: • Added information about commands for adding and removing items from collections • Added information about the set_timing_derate and report_skew commands • Added information about worst-case timing reporting • Minor editorial updates
November 2008	8.1.0	Updated for the Quartus Prime software version 8.1, including: • Added the following sections: "set_net_delay" on page 7–42 "Annotated Delay" on page 7–49 "report_net_delay" on page 7–66 • Updated the descriptions of the `-append` and `-file` <*name*> options in tables throughout the chapter • Updated entire chapter using 8½" × 11" chapter template • Minor editorial updates

Related Links

Altera Documentation Archive

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

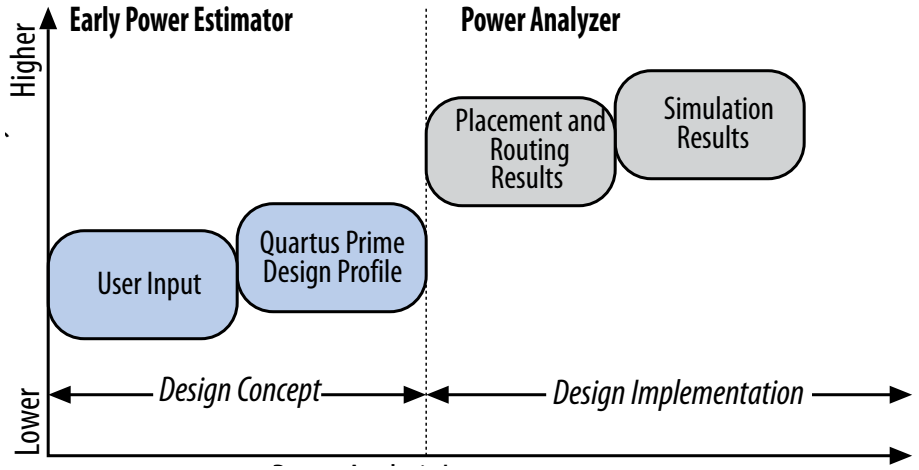
4 Power Analysis

The Quartus Prime Power Analysis tools allow you to estimate device power consumption accurately.

As designs grow larger and process technology continues to shrink, power becomes an increasingly important design consideration. When designing a PCB, you must estimate the power consumption of a device accurately to develop an appropriate power budget, and to design the power supplies, voltage regulators, heat sink, and cooling system.

The following figure shows the Power Analysis tools ability to estimate power consumption from early design concept through design implementation.

Figure 106. Power Analysis From Design Concept Through Design Implementation



For the majority of the designs, the Power Analyzer and the EPE spreadsheet have the following accuracy after the power models are final:

- Power Analyzer—±20% from silicon, assuming that the Power Analyzer uses the Value Change Dump File (`.vcd`) generated toggle rates.
- EPE spreadsheet— ±20% from the Power Analyzer results using `.vcd` generated toggle rates. 90% of EPE designs (using `.vcd` generated toggle rates exported from PPPA) are within ±30% silicon.

The toggle rates are derived using the Power Analyzer with a `.vcd` file generated from a gate level simulation representative of the system operation.

4.1 Types of Power Analyses

Understanding the uses of power analysis and the factors affecting power consumption helps you to use the Power Analyzer effectively. Power analysis meets the following significant planning requirements:

- **Thermal planning**—Thermal power is the power that dissipates as heat from the FPGA. You must use a heatsink or fan to act as a cooling solution for your device. The cooling solution must be sufficient to dissipate the heat that the device generates. The computed junction temperature must fall within normal device specifications.
- **Power supply planning**—Power supply is the power needed to run your device. Power supplies must provide adequate current to support device operation.

  Note: For power supply planning, use the EPE at the early stages of your design cycle. Use the Power Analyzer reports when your design is complete to get an estimate of your design power requirement.

The two types of analyses are closely related because much of the power supplied to the device dissipates as heat from the device; however, in some situations, the two types of analyses are not identical. For example, if you use terminated I/O standards, some of the power drawn from the power supply of the device dissipates in termination resistors rather than in the device.

Power analysis also addresses the activity of your design over time as a factor that impacts the power consumption of the device. The static power (P_{STATIC}) is the thermal power dissipated on chip, independent of user clocks. P_{STATIC} includes the leakage power from all FPGA functional blocks, except for I/O DC bias power and transceiver DC bias power, which are accounted for in the I/O and transceiver sections. Dynamic power is the additional power consumption of the device due to signal activity or toggling.

4.1.1 Differences between the EPE and the Quartus Prime Power Analyzer

The following table lists the differences between the EPE and the Quartus Prime Power Analyzer.

Table 23. Comparison of the EPE and Quartus Prime Power Analyzer

Characteristic	EPE	Quartus Prime Power Analyzer
Phase in the design cycle	Any time, but it is recommended to use Quartus Prime Power Analyzer for post-fit power analysis.	Post-fit
Tool requirements	Spreadsheet program	The Quartus Prime software
Accuracy	Medium	Medium to very high
Data inputs	• Resource usage estimates • Clock requirements • Environmental conditions • Toggle rate	• Post-fit design • Clock requirements • Signal activity defaults • Environmental conditions

continued...

Characteristic	EPE	Quartus Prime Power Analyzer
		• Register transfer level (RTL) simulation results (optional) • Post-fit simulation results (optional) • Signal activities per node or entity (optional)
Data outputs *(1)*	• Total thermal power dissipation • Thermal static power • Thermal dynamic power • Off-chip power dissipation • Current drawn from voltage supplies	• Total thermal power • Thermal static power • Thermal dynamic power • Thermal I/O power • Thermal power by design hierarchy • Thermal power by block type • Thermal power dissipation by clock domain • Off-chip (non-thermal) power dissipation • Device supply currents

The result of the Power Analyzer is only an estimation of power. Intel FPGA does not recommend using the result as a specification. The purpose of the estimation is to help you establish guidelines for the power budget of your design. It is important that you verify the actual power during device operation as the information is sensitive to the actual device design and the environmental operating conditions.

4.2 Factors Affecting Power Consumption

Understanding the following factors that affect power consumption allows you to use the Power Analyzer and interpret its results effectively:

- Device Selection
- Environmental Conditions
- Device Resource Usage
- Signal Activities

4.2.1 Device Selection

Device families have different power characteristics. Many parameters affect the device family power consumption, including choice of process technology, supply voltage, electrical design, and device architecture.

Power consumption also varies in a single device family. A larger device consumes more static power than a smaller device in the same family because of its larger transistor count. Dynamic power can also increase with device size in devices that employ global routing architectures.

5 EPE and Power Analyzer outputs vary by device family. For more information, refer to the device-specific Early Power Estimators (EPE) and Power Analyzer Page and Power Analyzer Reports in the Quartus Prime Help.

The choice of device package also affects the ability of the device to dissipate heat. This choice can impact your required cooling solution choice to comply to junction temperature constraints.

Process variation can affect power consumption. Process variation primarily impacts static power because sub-threshold leakage current varies exponentially with changes in transistor threshold voltage. Therefore, you must consult device specifications for static power and not rely on empirical observation. Process variation has a weak effect on dynamic power.

4.2.2 Environmental Conditions

Operating temperature primarily affects device static power consumption. Higher junction temperatures result in higher static power consumption. The device thermal power and cooling solution that you use must result in the device junction temperature remaining within the maximum operating range for the device. The main environmental parameters affecting junction temperature are the cooling solution and ambient temperature.

The following table lists the environmental conditions that could affect power consumption.

Table 24. Environmental Conditions that Could Affect Power Consumption

Environmental Conditions	Description
Airflow	A measure of how quickly the device removes heated air from the vicinity of the device and replaces it with air at ambient temperature. You can either specify airflow as "still air" when you are not using a fan, or as the linear feet per minute rating of the fan in the system. Higher airflow decreases thermal resistance.
Heat Sink and Thermal Compound	A heat sink allows more efficient heat transfer from the device to the surrounding area because of its large surface area exposed to the air. The thermal compound that interfaces the heat sink to the device also influences the rate of heat dissipation. The case-to-ambient thermal resistance (θ_{CA}) parameter describes the cooling capacity of the heat sink and thermal compound employed at a given airflow. Larger heat sinks and more effective thermal compounds reduce θ_{CA}.
Junction Temperature	The junction temperature of a device is equal to: $T_{Junction} = T_{Ambient} + P_{Thermal} \cdot \theta_{JA}$ in which θ_{JA} is the total thermal resistance from the device transistors to the environment, having units of degrees Celsius per watt. The value θ_{JA} is equal to the sum of the junction-to-case (package) thermal resistance (θ_{JC}), and the case-to-ambient thermal resistance (θ_{CA}) of your cooling solution.
Board Thermal Model	The junction-to-board thermal resistance (θ_{JB}) is the thermal resistance of the path through the board, having units of degrees Celsius per watt. To compute junction temperature, you can use this board thermal model along with the board temperature, the top-of-chip θ_{JA} and ambient temperatures.

4.2.3 Device Resource Usage

The number and types of device resources used greatly affects power consumption.

- **Number, Type, and Loading of I/O Pins**—Output pins drive off-chip components, resulting in high-load capacitance that leads to a high-dynamic power per transition. Terminated I/O standards require external resistors that draw constant (static) power from the output pin.
- **Number and Type of Hard Logic Blocks**—A design with more logic elements (LEs), multiplier elements, memory blocks, transceiver blocks or HPS system tends to consume more power than a design with fewer circuit elements. The operating mode of each circuit element also affects its power consumption. For example, a DSP block performing 18 × 18 multiplications and a DSP block performing multiply-accumulate operations consume different amounts of dynamic power because of different amounts of charging internal capacitance on each transition. The operating mode of a circuit element also affects static power.
- **Number and Type of Global Signals**—Global signal networks span large portions of the device and have high capacitance, resulting in significant dynamic power consumption. The type of global signal is important as well. For example, Stratix V devices support global clocks and quadrant (regional) clocks. Global clocks cover the entire device, whereas quadrant clocks only span one-fourth of the device. Clock networks that span smaller regions have lower capacitance and tend to consume less power. The location of the logic array blocks (LABs) driven by the clock network can also have an impact because the Quartus Prime software automatically disables unused branches of a clock.

4.2.4 Signal Activities

The behavior of each signal in your design is an important factor in estimating power consumption. The following table lists the two vital behaviors of a signal, which are toggle rate and static probability:

Table 25. Signal Behavior

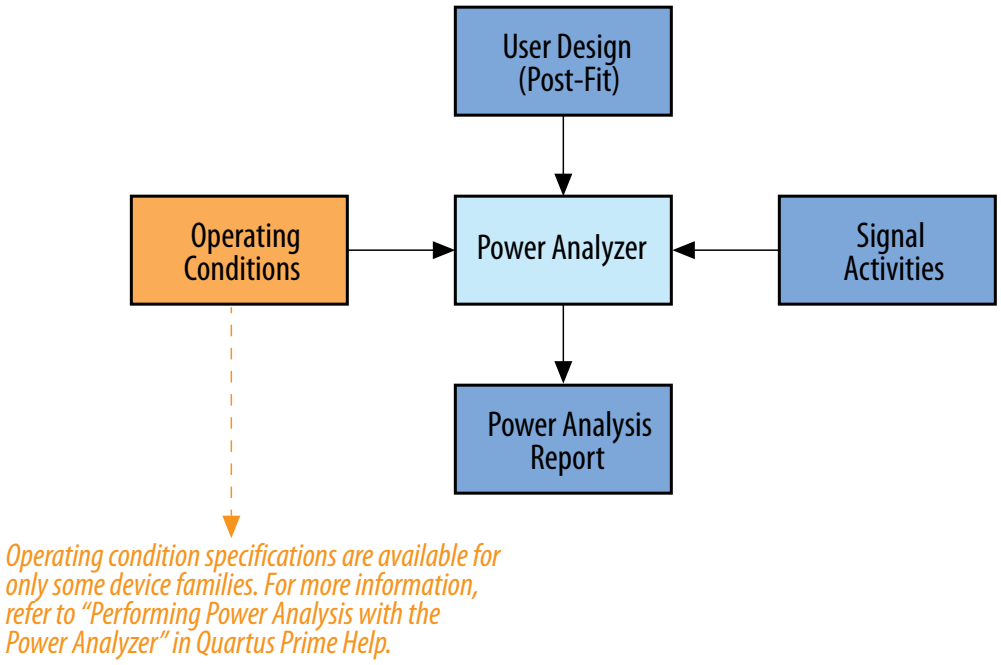
Signal Behavior	Description
Toggle rate	• The toggle rate of a signal is the average number of times that the signal changes value per unit of time. The units for toggle rate are transitions per second and a transition is a change from 1 to 0, or 0 to 1. • Dynamic power increases linearly with the toggle rate as you charge the board trace model more frequently for logic and routing. The Quartus Prime software models full rail-to-rail switching. For high toggle rates, especially on circuit output I/O pins, the circuit can transition before fully charging the downstream capacitance. The result is a slightly conservative prediction of power by the Power Analyzer.
Static probability	• The static probability of a signal is the fraction of time that the signal is logic 1 during the period of device operation that is being analyzed. Static probability ranges from 0 (always at ground) to 1 (always at logic-high). • Static probabilities of their input signals can sometimes affect the static power that routing and logic consume. This effect is due to state-dependent leakage and has a larger effect on smaller process geometries. The Quartus Prime software models this effect on devices at 90 nm or smaller if it is important to the power estimate. The static power also varies with the static probability of a logic 1 or 0 on the I/O pin when output I/O standards drive termination resistors.

Note: To get accurate results from the power analysis, the signal activities for analysis must represent the actual operating behavior of your design. Inaccurate signal toggle rate data is the largest source of power estimation error.

4.3 Power Analyzer Flow

The Power Analyzer supports accurate power estimations by allowing you to specify the important design factors affecting power consumption. The following figure shows the high-level Power Analyzer flow.

Figure 107. Power Analyzer High-Level Flow



To obtain accurate I/O power estimates, the Power Analyzer requires you to synthesize your design and then fit your design to the target device. You must specify the electrical standard on each I/O cell and the board trace model on each I/O standard in your design.

4.3.1 Operating Settings and Conditions

You can specify device power characteristics, operating voltage conditions, and operating temperature conditions for power analysis in the Quartus Prime software.

On the **Operating Settings and Conditions** page of the **Settings** dialog box, you can specify whether the device has typical power consumption characteristics or maximum power consumption characteristics.

On the **Voltage** page of the **Settings** dialog box, you can view the operating voltage conditions for each power rail in the device, and specify supply voltages for power rails with selectable supply voltages.

Note: The Quartus Prime Fitter may override some of the supply voltages settings specified in this chapter. For example, supply voltages for several Stratix V transceiver power supplies depend on the data rate used. If the Fitter detects that voltage required is different from the one specified in the **Voltage** page, it will automatically set the correct voltage for relevant rails. The Quartus Prime Power Analyzer uses voltages selected by the Fitter if they conflict with the settings specified in the **Voltage** page.

On the **Temperature** page of the **Settings** dialog box, you can specify the thermal operating conditions of the device.

Related Links

- Operating Settings and Conditions Page (Settings Dialog Box)
- Voltage Page (Settings Dialog Box)
- Temperature Page (Settings Dialog Box)

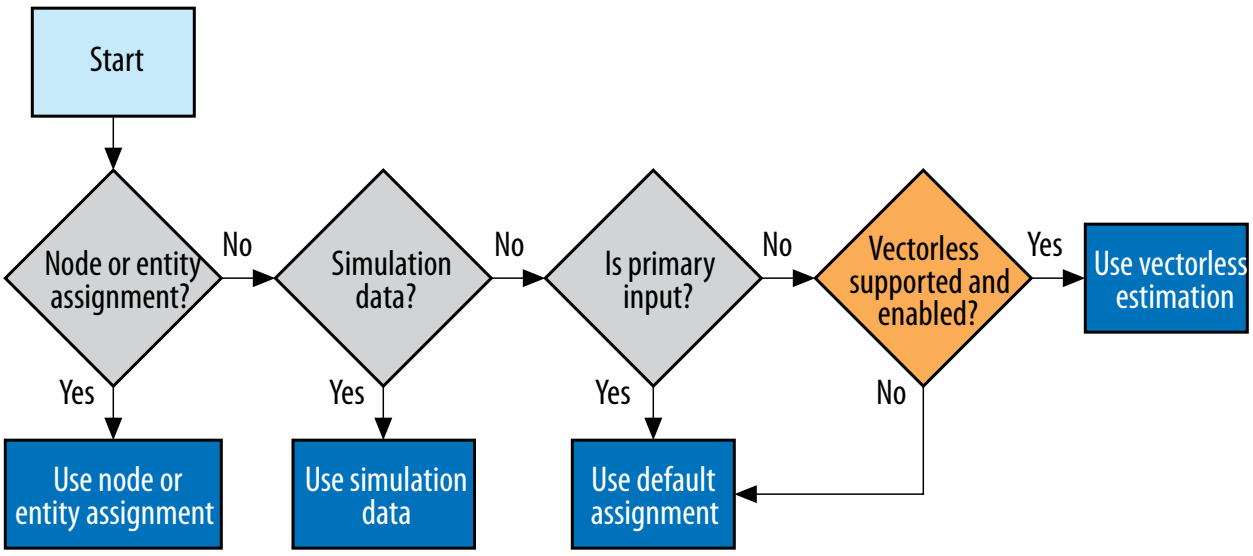
4.3.2 Signal Activities Data Sources

The Power Analyzer provides a flexible framework for specifying signal activities. The framework reflects the importance of using representative signal-activity data during power analysis. Use the following sources to provide information about signal activity:

- Simulation results
- User-entered node, entity, and clock assignments
- User-entered default toggle rate assignment
- Vectorless estimation

The Power Analyzer allows you to mix and match the signal-activity data sources on a signal-by-signal basis. The following figure shows the priority scheme applied to each signal.

Figure 108. Signal-Activity Data Source Priority Scheme



4.3.2.1 Simulation Results

The Power Analyzer directly reads the waveforms generated by a design simulation. Static probability and toggle rate can be calculated for each signal from the simulation waveform. Power analysis is most accurate when you use representative input stimuli to generate simulations.

The Power Analyzer reads results generated by the following simulators:

- ModelSim
- ModelSim - Intel FPGA Edition
- QuestaSim
- Active-HDL
- NCSim
- VCS
- VCS MX
- Riviera-PRO

Signal activity and static probability information are derived from a Verilog Value Change Dump File (`.vcd`). For more information, refer to Signal Activities on page 130.

For third-party simulators, use the EDA `Tool Settings` to specify the Generate Value Change Dump (VCD) file script option in the Simulation page of the Settings dialog box. These scripts instruct the third-party simulators to generate a `.vcd` that encodes the simulated waveforms. The Quartus Prime Power Analyzer reads this file directly to derive the toggle rate and static probability data for each signal.

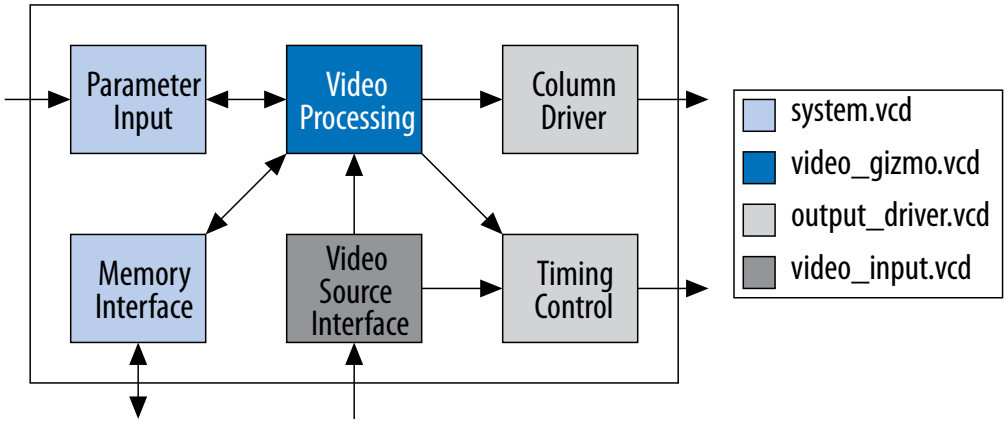
Third-party EDA simulators, other than those listed, can generate a `.vcd` that you can use with the Power Analyzer. For those simulators, you must manually create a simulation script to generate the appropriate `.vcd`.

Note: You can use a `.vcd` created for power analysis to optimize your design for power during fitting by utilizing the appropriate settings in the power optimization list, available from **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Fitter)**.

4.4 Using Simulation Files in Modular Design Flows

A common design practice is to create modular or hierarchical designs in which you develop each design entity separately, and then instantiate these modules in a higher-level entity to form a complete design. You can perform simulation on a complete design or on each module for verification. The Power Analyzer supports modular design flows when reading the signal activities from simulation files. The following figure shows an example of a modular design flow.

Figure 109. Modular Simulation Flow



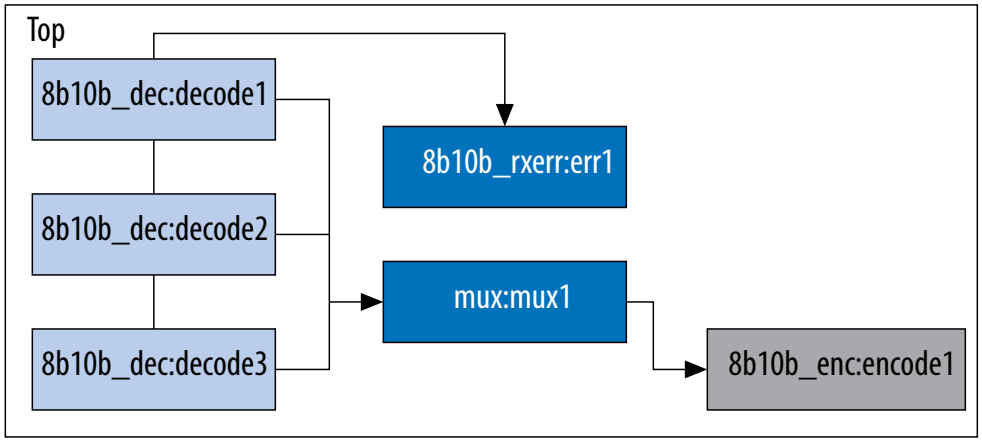
When specifying a simulation file (a `.vcd`), the software provides support to specify an associated design entity name, such that the Power Analyzer imports the signal activities derived from that file for the specified design entity. The Power Analyzer also supports the specification of multiple `.vcd` files for power analysis, with each having an associated design entity name to enable the integration of partial design simulations into a complete design power analysis. When specifying multiple `.vcd` files for your design, more than one simulation file can contain signal-activity information for the same signal.

Note: When you apply multiple `.vcd` files to the same design entity, the signal activity used in the power analysis is the equal-weight arithmetic average of each `.vcd`.

Note: When you apply multiple simulation files to design entities at different levels in your design hierarchy, the signal activity in the power analysis derives from the simulation file that applies to the most specific design entity.

The following figure shows an example of a hierarchical design. The top-level module of your design, called `Top`, consists of three 8b/10b decoders, followed by a `mux`. The software then encodes the output of the `mux` to produce the final output of the top-level module. An error-handling module handles any 8b/10b decoding errors. The Top module contains the top-level entity of your design and any logic not defined as part of another module. The design file for the top-level module might be a wrapper for the hierarchical entities below it, or it might contain its own logic. The following usage scenarios show common ways that you can simulate your design and import the `.vcd` into the Power Analyzer.

Figure 110. Example Hierarchical Design



4.4.1 Complete Design Simulation

You can simulate the entire design and generate a `.vcd` from a third-party simulator. The Power Analyzer can then import the `.vcd` (specifying the top-level design). The resulting power analysis uses the signal activities information from the generated `.vcd`, including those that apply to submodules, such as `decode [1-3]`, `err1`, `mux1`, and `encode1`.

4.4.2 Modular Design Simulation

You can independently simulate of the top-level design, and then import all the resulting `.vcd` files into the Power Analyzer. For example, you can simulate the `8b10b_dec` independent of the entire design and `mux`, `8b10b_rxerr`, and `8b10b_enc`. You can then import the `.vcd` files generated from each simulation by specifying the appropriate instance name. For example, if the files produced by the simulations are `8b10b_dec.vcd`, `8b10b_enc.vcd`, `8b10b_rxerr.vcd`, and `mux.vcd`, you can use the import specifications in the following table:

Table 26. Import Specifications

File Name	Entity
`8b10b_dec.vcd`	`Top|8b10b_dec:decode1`
`8b10b_dec.vcd`	`Top|8b10b_dec:decode2`
`8b10b_dec.vcd`	`Top|8b10b_dec:decode3`
`8b10b_rxerr.vcd`	`Top|8b10b_rxerr:err1`
`8b10b_enc.vcd`	`Top|8b10b_enc:encode1`
`mux.vcd`	`Top|mux:mux1`

The resulting power analysis applies the simulation vectors in each file to the assigned entity. Simulation provides signal activities for the pins and for the outputs of functional blocks. If the inputs to an entity instance are input pins for the entire

design, the simulation file associated with that instance does not provide signal activities for the inputs of that instance. For example, an input to an entity such as `mux1` has its signal activity specified at the output of one of the decode entities.

4.4.3 Multiple Simulations on the Same Entity

You can perform multiple simulations of an entire design or specific modules of a design. For example, in the process of verifying the top-level design, you can have three different simulation testbenches: one for normal operation, and two for corner cases. Each of these simulations produces a separate `.vcd`. In this case, apply the different `.vcd` file names to the same top-level entity, as shown in the following table.

Table 27. Multiple Simulation File Names and Entities

File Name	Entity
`normal.vcd`	`Top`
`corner1.vcd`	`Top`
`corner2.vcd`	`Top`

The resulting power analysis uses an arithmetic average of the signal activities calculated from each simulation file to obtain the final signal activities used. If a signal `err_out` has a toggle rate of zero transition per second in `normal.vcd`, 50 transitions per second in `corner1.vcd`, and 70 transitions per second in `corner2.vcd`, the final toggle rate in the power analysis is 40 transitions per second.

If you do not want the Power Analyzer to read information from multiple instances and take an arithmetic average of the signal activities, use a `.vcd` that includes only signals from the instance that you care about.

4.4.4 Overlapping Simulations

You can perform a simulation on the entire design, and more exhaustive simulations on a submodule, such as `8b10b_rxerr`. The following table lists the import specification for overlapping simulations.

Table 28. Overlapping Simulation Import Specifications

File Name	Entity
`full_design.vcd`	`Top`
`error_cases.vcd`	`Top|8b10b_rxerr:err1`

In this case, the software uses signal activities from `error_cases.vcd` for all the nodes in the generated `.vcd` and uses signal activities from `full_design.vcd` for only those nodes that do not overlap with nodes in `error_cases.vcd`. In general, the more specific hierarchy (the most bottom-level module) derives signal activities for overlapping nodes.

4.4.5 Partial Simulations

You can perform a simulation in which the entire simulation time is not applicable to signal-activity calculation. For example, if you run a simulation for 10,000 clock cycles and reset the chip for the first 2,000 clock cycles. If the Power Analyzer performs the

signal-activity calculation over all 10,000 cycles, the toggle rates are only 80% of their steady state value (because the chip is in reset for the first 20% of the simulation). In this case, you must specify the useful parts of the `.vcd` for power analysis. The **Limit VCD Period** option enables you to specify a start and end time when performing signal-activity calculations.

4.4.5.1 Specifying Start and End Time when Performing Signal-Activity Calculations using the Limit VCD Period Option

To specify a start and end time when performing signal-activity calculations using the **Limit VCD period** option, follow these steps:

1. In the Quartus Prime software, on the Assignments menu, click **Settings**.
2. Under the Category list, click **Power Analyzer Settings**.
3. Turn on the **Use input file(s) to initialize toggle rates and static probabilities during power analysis** option.
4. Click **Add**.
5. In the **File name** and **Entity** fields, browse to the necessary files.
6. Under **Simulation period**, turn on **VCD file** and **Limit VCD period** options.
7. In the **Start time** and **End time** fields, specify the desired start and end time.
8. Click **OK**.

You can also use the following tcl or qsf assignment to specify `.vcd` files:

```
set_global_assignment -name POWER_INPUT_FILE_NAME "test.vcd" -section_id test.vcd
set_global_assignment -name POWER_INPUT_FILE_TYPE VCD -section_id test.vcd
set_global_assignment -name POWER_VCD_FILE_START_TIME "10 ns" -section_id test.vcd
set_global_assignment -name POWER_VCD_FILE_END_TIME "1000 ns" -section_id test.vcd
set_instance_assignment -name POWER_READ_INPUT_FILE test.vcd -to test_design
```

Related Links

- set_power_file_assignment
- Add/Edit Power Input File Dialog Box

4.4.6 Node Name Matching Considerations

Node name mismatches happen when you have `.vcd` applied to entities other than the top-level entity. In a modular design flow, the gate-level simulation files created in different Quartus Prime projects might not match their node names with the current Quartus Prime project.

For example, you may have a file named `8b10b_enc.vcd`, which the Quartus Prime software generates in a separate project called `8b10b_enc` while simulating the `8b10b` encoder. If you import the `.vcd` into another project called `Top`, you might encounter name mismatches when applying the `.vcd` to the `8b10b_enc` module in the `Top` project. This mismatch happens because the Quartus Prime software might name all the combinational nodes in the `8b10b_enc.vcd` differently than in the `Top` project.

4.4.7 Glitch Filtering

The Power Analyzer defines a glitch as two signal transitions so closely spaced in time that the pulse, or glitch, occurs faster than the logic and routing circuitry can respond. The output of a transport delay model simulator contains glitches for some signals. The logic and routing structures of the device form a low-pass filter that filters out glitches that are tens to hundreds of picoseconds long, depending on the device family.

Some third-party simulators use different models than the transport delay model as the default model. Different models cause differences in signal activity and power estimation. The inertial delay model, which is the ModelSim default model, filters out more glitches than the transport delay model and usually yields a lower power estimate.

Note: Intel FPGA recommends that you use the transport simulation model when using the Quartus Prime software glitch filtering support with third-party simulators. Simulation glitch filtering has little effect if you use the inertial simulation model.

Glitch filtering in a simulator can also filter a glitch on one logic element (LE) (or other circuit element) output from propagating to downstream circuit elements to ensure that the glitch does not affect simulated results. Glitch filtering prevents a glitch on one signal from producing non-physical glitches on all downstream logic, which can result in a signal toggle rate and a power estimate that are too high. Circuit elements in which every input transition produces an output transition, including multipliers and logic cells configured to implement XOR functions, are especially prone to glitches. Therefore, circuits with such functions can have power estimates that are too high when glitch filtering is not used.

Note: Intel FPGA recommends that you use the glitch filtering feature to obtain the most accurate power estimates. For `.vcd` files, the Power Analyzer flows support two levels of glitch filtering.

4.4.7.1 Enabling First Level of Glitch Filtering

To enable the first level of glitch filtering in the Quartus Prime software for supported third-party simulators, follow these steps:

1. On the Assignments menu, click **Settings**.
2. In the **Category** list, select **Simulation under EDA Tool Settings**.
3. Select the **Tool name** to use for the simulation.
4. Turn on **Enable glitch filtering**.

4.4.7.2 Enabling Second Level of Glitch Filtering

The second level of glitch filtering occurs while the Power Analyzer is reading the .vcd generated by a third-party simulator. To enable the second level of glitch filtering, follow these steps:

1. On the Assignments menu, click **Settings**.
2. In the **Category** list, select **Power Analyzer Settings**.
3. Under **Input File(s)**, turn on **Perform glitch filtering on VCD files**.

The `.vcd` file reader performs filtering complementary to the filtering performed during simulation and is often not as effective. While the `.vcd` file reader can remove glitches on logic blocks, the file reader cannot determine how a given glitch affects downstream logic and routing, and may eliminate the impact of the glitch completely. Filtering the glitches during simulation avoids switching downstream routing and logic automatically.

Note: When running simulation for design verification (rather than to produce input to the Power Analyzer), Intel FPGA recommends that you turn off the glitch filtering option to produce the most rigorous and conservative simulation from a functionality viewpoint. When performing simulation to produce input for the Power Analyzer, Intel FPGA recommends that you turn on the glitch filtering to produce the most accurate power estimates.

4.4.8 Node and Entity Assignments

You can assign toggle rates and static probabilities to individual nodes and entities in the design. These assignments have the highest priority, overriding data from all other signal-activity sources.

You must use the Assignment Editor or Tcl commands to create the **Power Toggle Rate** and **Power Static Probability** assignments. You can specify the power toggle rate as an absolute toggle rate in transitions per second using the **Power Toggle Rate** assignment, or you can use the **Power Toggle Rate Percentage** assignment to specify a toggle rate relative to the clock domain of the assigned node for a more specific assignment made in terms of hierarchy level.

Note: If you use the **Power Toggle Rate Percentage** assignment, and the node does not have a clock domain, the Quartus Prime software issues a warning and ignores the assignment.

Assigning toggle rates and static probabilities to individual nodes and entities is appropriate for signals in which you have knowledge of the signal or entity being analyzed. For example, if you know that a 100 MHz data bus or memory output produces data that is essentially random (uncorrelated in time), you can directly enter a 0.5 static probability and a toggle rate of 50 million transitions per second.

The Power Analyzer treats bidirectional I/O pins differently. The combinational input port and the output pad for a pin share the same name. However, those ports might not share the same signal activities. For reading signal-activity assignments, the Power Analyzer creates a distinct name <*node_name~output*> when configuring the bidirectional signal as an output and <*node_name~result*> when configuring the signal as an input. For example, if a design has a bidirectional pin named `MYPIN`, assignments for the combinational input use the name `MYPIN~result`, and the assignments for the output pad use the name `MYPIN~output`.

Note: When you create the logic assignment in the Assignment Editor, you cannot find the `MYPIN~result` and `MYPIN~output` node names in the Node Finder. Therefore, to create the logic assignment, you must manually enter the two differentiating node names to create the assignment for the input and output port of the bidirectional pin.

4.4.8.1 Timing Assignments to Clock Nodes

For clock nodes, the Power Analyzer uses timing requirements to derive the toggle rate when neither simulation data nor user-entered signal-activity data is available. f_{MAX} requirements specify full cycles per second, but each cycle represents a rising transition and a falling transition. For example, a clock f_{MAX} requirement of 100 MHz corresponds to 200 million transitions per second for the clock node.

4.4.9 Default Toggle Rate Assignment

You can specify a default toggle rate for primary inputs and other nodes in your design. The Power Analyzer uses the default toggle rate when no other method specifies the signal-activity data.

The Power Analyzer specifies the toggle rate in absolute terms (transitions per second), or as a fraction of the clock rate in effect for each node. The toggle rate for a clock derives from the timing settings for the clock. For example, if the Power Analyzer specifies a clock with an f_{MAX} constraint of 100 MHz and a default relative toggle rate of 20%, nodes in this clock domain transition in 20% of the clock periods, or 20 million transitions occur per second. In some cases, the Power Analyzer cannot determine the clock domain for a node because either the Power Analyzer cannot determine a clock domain for the node, or the clock domain is ambiguous. For example, the Power Analyzer may not be able to determine a clock domain for a node if the user did not specify sufficient timing assignments. In these cases, the Power Analyzer substitutes and reports a toggle rate of zero.

4.4.10 Vectorless Estimation

For some device families, the Power Analyzer automatically derives estimates for signal activity on nodes with no simulation or user-entered signal-activity data. Vectorless estimation statistically estimates the signal activity of a node based on the signal activities of nodes feeding that node, and on the actual logic function that the node implements. Vectorless estimation cannot derive signal activities for primary inputs. Vectorless estimation is accurate for combinational nodes, but not for registered nodes. Therefore, the Power Analyzer requires simulation data for at least the registered nodes and I/O nodes for accuracy.

The **Power Analyzer Settings** dialog box allows you to disable vectorless estimation. When turned on, vectorless estimation takes precedence over default toggle rates. Vectorless estimation does not override clock assignments.

To disable vectorless estimation, perform the following steps:

1. In the Quartus Prime software, on the Assignments menu, click **Settings**.
2. In the Category list, select **Power Analyzer Settings**.
3. Turn off the **Use vectorless estimation** option.

4.5 Using the Power Analyzer

For flows that use the Power Analyzer, you must first synthesize your design, and then fit it to the target device. You must either provide timing assignments for all the clocks in your design, or use a simulation-based flow to generate activity data. You must specify the I/O standard on each device input and output and the board trace model on each output in your design.

4.5.1 Common Analysis Flows

You can use the analysis flows in this section with the Power Analyzer. However, vectorless activity estimation is only available for some device families.

4.5.1.1 Signal Activities from RTL (Functional) Simulation, Supplemented by Vectorless Estimation

In the functional simulation flow, simulation provides toggle rates and static probabilities for all pins and registers in your design. Vectorless estimation fills in the values for all the combinational nodes between pins and registers, giving good results. This flow usually provides a compilation time benefit when you use the third-party RTL simulator.

4.5.1.1.1 RTL Simulation Limitation

RTL simulation may not provide signal activities for all registers in the post-fitting netlist because synthesis loses some register names. For example, synthesis might automatically transform state machines and counters, thus changing the names of registers in those structures.

4.5.1.2 Signal Activities from Vectorless Estimation and User-Supplied Input Pin Activities

The vectorless estimation flow provides a low level of accuracy, because vectorless estimation for registers is not entirely accurate.

4.5.1.3 Signal Activities from User Defaults Only

The user defaults only flow provides the lowest degree of accuracy.

4.5.2 Using .vcd for Power Estimation

Use a `.vcd` generated by your gate-level timing simulation tool as the source of activity data for accurate power estimation. The gate-level timing simulation `.vcd` includes all the routing resources and the exact logic array resource usage. Follow the documentation for your simulation tool to generate a `.vcd` during simulation. Specify the `.vcd` as the input to the Power Analyzer to estimate power for your design.

4.6 Power Analyzer Compilation Report

The following table list the items in the Compilation Report of the Power Analyzer section.

Section	Description
Summary	The Summary section of the report shows the estimated total thermal power consumption of your design. This includes dynamic, static, and I/O thermal power consumption. The I/O thermal power includes the total I/O power drawn from the V_{CCIO} and V_{CCPD} power supplies and the power drawn from V_{CCINT} in the I/O subsystem including I/O buffers and I/O registers. The report also includes a confidence metric that reflects the overall quality of the data sources for the signal activities. For example, a **Low** power estimation confidence value reflects that you have provided insufficient

continued...

Section	Description
	toggle rate data, or most of the signal-activity information used for power estimation is from default or vectorless estimation settings. For more information about the input data, refer to the Power Analyzer Confidence Metric report.
Settings	The Settings section of the report shows the Power Analyzer settings information of your design, including the default input toggle rates, operating conditions, and other relevant setting information.
Simulation Files Read	The Simulation Files Read section of the report lists the simulation output file that the `.vcd` used for power estimation. This section also includes the file ID, file type, entity, VCD start time, VCD end time, the unknown percentage, and the toggle percentage. The unknown percentage indicates the portion of the design module unused by the simulation vectors.
Operating Conditions Used	The Operating Conditions Used section of the report shows device characteristics, voltages, temperature, and cooling solution, if any, during the power estimation. This section also shows the entered junction temperature or auto-computed junction temperature during the power analysis.
Thermal Power Dissipated by Block	The Thermal Power Dissipated by Block section of the report shows estimated thermal dynamic power and thermal static power consumption categorized by atoms. This information provides you with estimated power consumption for each atom in your design. By default, this section does not contain any data, but you can turn on the report with the **Write power dissipation by block to report file** option on the **Power Analyzer Settings** page.
Thermal Power Dissipation by Block Type (Device Resource Type)	This Thermal Power Dissipation by Block Type (Device Resource Type) section of the report shows the estimated thermal dynamic power and thermal static power consumption categorized by block types. This information is further categorized by estimated dynamic and static power and provides an average toggle rate by block type. Thermal power is the power dissipated as heat from the FPGA device.
Thermal Power Dissipation by Hierarchy	This Thermal Power Dissipation by Hierarchy section of the report shows estimated thermal dynamic power and thermal static power consumption categorized by design hierarchy. This information is further categorized by the dynamic and static power that was used by the blocks and routing in that hierarchy. This information is useful when locating modules with high power consumption in your design.
Core Dynamic Thermal Power Dissipation by Clock Domain	The Core Dynamic Thermal Power Dissipation by Clock Domain section of the report shows the estimated total core dynamic power dissipation by each clock domain, which provides designs with estimated power consumption for each clock domain in the design. If the clock frequency for a domain is unspecified by a constraint, the clock frequency is listed as "unspecified." For all the combinational logic, the clock domain is listed as no clock with zero MHz.
Current Drawn from Voltage Supplies	The Current Drawn from Voltage Supplies section of the report lists the current drawn from each voltage supply. The V_{CCIO} and V_{CCPD} voltage supplies are further categorized by I/O bank and by voltage. This section also lists the minimum safe power supply size (current supply ability) for each supply voltage. Minimum current requirement can be higher than user mode current requirement in cases in which the supply has a specific power up current requirement that goes beyond user mode requirement, such as the V_{CCPD} power rail in Stratix III and Stratix IV devices, and the V_{CCIO} power rail in Stratix IV devices. The I/O thermal power dissipation on the summary page does not correlate directly to the power drawn from the V_{CCIO} and V_{CCPD} voltage supplies listed in this report. This is because the I/O thermal power dissipation value also includes portions of the V_{CCINT} power, such as the I/O element (IOE) registers, which are modeled as I/O power, but do not draw from the V_{CCIO} and V_{CCPD} supplies. The reported current drawn from the I/O Voltage Supplies (ICCIO and ICCPD) as reported in the Power Analyzer report includes any current drawn through the I/O into off-chip termination resistors. This can result in ICCIO and ICCPD values that are higher than the reported I/O thermal power, because this off-chip current dissipates as heat elsewhere and does not factor in the calculation of device temperature. Therefore, total I/O thermal power does not equal the sum of current drawn from each V_{CCIO} and V_{CCPD} supply multiplied by V_{CCIO} and V_{CCPD} voltage. For SoC devices or for Arria V SoC and Cyclone® V SoC devices, there is no standalone ICC_AUX_SHARED current drawn information. The ICC_AUX_SHARED is reported together with ICC_AUX.
Confidence Metric Details	The Confidence Metric is defined in terms of the total weight of signal activity data sources for both combinational and registered signals. Each signal has two data sources allocated to it; a toggle rate source and a static probability source.

continued...

Section	Description
	The Confidence Metric Details section also indicates the quality of the signal toggle rate data to compute a power estimate. The confidence metric is low if the signal toggle rate data comes from poor predictors of real signal toggle rates in the device during an operation. Toggle rate data that comes from simulation, user-entered assignments on specific signals or entities are reliable. Toggle rate data from default toggle rates (for example, 12.5% of the clock period) or vectorless estimation are relatively inaccurate. This section gives an overall confidence rating in the toggle rate data, from low to high. This section also summarizes how many pins, registers, and combinational nodes obtained their toggle rates from each of simulation, user entry, vectorless estimation, or default toggle rate estimations. This detailed information helps you understand how to increase the confidence metric, letting you determine your own confidence in the toggle rate data.
Signal Activities	The Signal Activities section lists toggle rates and static probabilities assumed by power analysis for all signals with fan-out and pins. This section also lists the signal type (pin, registered, or combinational) and the data source for the toggle rate and static probability. By default, this section does not contain any data, but you can turn on the report with the **Write signal activities to report file** option on the **Power Analyzer Settings** page. Intel FPGA recommends that you keep the **Write signal activities to report file** option turned off for a large design because of the large number of signals present. You can use the Assignment Editor to specify that activities for individual nodes or entities are reported by assigning an on value to those nodes for the **Power Report Signal Activities** assignment.
Messages	The Messages section lists the messages that the Quartus Prime software generates during the analysis.

4.7 Scripting Support

You can run procedures and create settings described in this chapter in a Tcl script. You can also run some procedures at a command prompt. For more information about scripting command options, refer to the Quartus Prime Command-Line and Tcl API Help browser. To run the Help browser, type the following command at the command prompt:

```
quartus_sh --qhelp ←
```

Related Links

- Tcl Scripting
- API Functions for Tcl
- Quartus Prime Settings File Reference Manual

4.7.1 Running the Power Analyzer from the Command–Line

The executable to run the Power Analyzer is `quartus_pow`. For a complete listing of all command–line options supported by `quartus_pow`, type the following command at a system command prompt:

```
quartus_pow --help ←
```

or-

```
quartus_sh --qhelp ←
```

The following lists the examples of using the `quartus_pow` executable. Type the command listed in the following section at a system command prompt. These examples assume that operations are performed on Quartus Prime project called *sample*.

To instruct the Power Analyzer to generate a EPE File:

```
quartus_pow sample --output_epe=sample.csv ←
```

To instruct the Power Analyzer to generate a EPE File without performing the power estimate:

```
quartus_pow sample --output_epe=sample.csv --estimate_power=off  ←
```

To instruct the Power Analyzer to use a .vcd as input (sample.vcd):

```
quartus_pow sample --input_vcd=sample.vcd ←
```

To instruct the Power Analyzer to use two .vcd files as input files (sample1.vcd and sample2.vcd), perform glitch filtering on the .vcd and use a default input I/O toggle rate of 10,000 transitions per second:

```
quartus_pow sample --input_vcd=sample1.vcd --input_vcd=sample2.vcd \
--vcd_filter_glitches=on --\
default_input_io_toggle_rate=10000transitions/s ←
```

To instruct the Power Analyzer to not use an input file, a default input I/O toggle rate of 60%, no vectorless estimation, and a default toggle rate of 20% on all remaining signals:

```
quartus_pow sample --no_input_file --default_input_io_toggle_rate=60% \
--use_vectorless_estimation=off --default_toggle_rate=20% ←
```

Note: No command–line options are available to specify the information found on the **Power Analyzer Settings Operating Conditions** page. Use the Quartus Prime GUI to specify these options.

The `quartus_pow` executable creates a report file, *<revision name>*`.pow.rpt`. You can locate the report file in the main project directory. The report file contains the same information in Power Analyzer Compilation Report on page 141.

4.8 Document Revision History

The following table lists the revision history for this chapter.

Date	Version	Changes
2017.05.08	17.0.0	Removed references to PowerPlay® name. Power analysis occurs in the Quartus Prime Power Analyzer.
2016.10.31	16.1.0	• Implemented Intel rebranding. • Removed support for `.vcd` generation by the Compiler. Generate `.vcd` files for power estimation in your EDA simmulator.
2015.11.02	15.1.0	Changed instances of *Quartus II* to *Quartus Prime*.

continued...

Date	Version	Changes
2014.12.15	14.1.0	• Removed Signal Activities from Full Post-Fit Netlist (Timing) Simulation and Signal Activities from Full Post-Fit Netlist (Zero Delay) Simulation sections as these are no longer supported. • Updated location of Fitter Settings, Analysis & Synthesis Settings, and Physical Synthesis Optimizations to Compiler Settings.
2014.08.18	14.0a10.0	Updated "Current Drawn from Voltage Supplies" to clarify that for SoC devices or for Arria V SoC and Cyclone V SoC devices, there is no standalone ICC_AUX_SHARED current drawn information. The ICC_AUX_SHARED is reported together with ICC_AUX.
November 2012	12.1.0	• Updated "Types of Power Analyses" on page 8–2, and "Confidence Metric Details" on page 8–23. • Added "Importance of .vcd" on page 8–20, and "Avoiding Power Estimation and Hardware Measurement Mismatch" on page 8–24
June 2012	12.0.0	• Updated "Current Drawn from Voltage Supplies" on page 8–22. • Added "Using the HPS Power Calculator" on page 8–7.
November 2011	10.1.1	• Template update. • Minor editorial updates.
December 2010	10.1.0	• Added links to Quartus Prime Help, removed redundant material. • Moved "Creating PowerPlay EPE Spreadsheets" to page 8–6. • Minor edits.
July 2010	10.0.0	• Removed references to the Quartus Prime Simulator. • Updated Table 8–1 on page 8–6, Table 8–2 on page 8–13, and Table 8–3 on page 8–14. • Updated Figure 8–3 on page 8–9, Figure 8–4 on page 8–10, and Figure 8–5 on page 8–12.
November 2009	9.1.0	• Updated "Creating PowerPlay EPE Spreadsheets" on page 8–6 and "Simulation Results" on page 8–10. • Added "Signal Activities from Full Post-Fit Netlist (Zero Delay) Simulation" on page 8–19 and "Generating a .vcd from Full Post-Fit Netlist (Zero Delay) Simulation" on page 8–21. • Minor changes to "Generating a .vcd from ModelSim Software" on page 8–21. • Updated Figure 11–8 on page 11–24.
March 2009	9.0.0	• This chapter was chapter 11 in version 8.1. • Removed Figures 11-10, 11-11, 11-13, 11-14, and 11-17 from 8.1 version.
November 2008	8.1.0	• Updated for the Quartus Prime software version 8.1. • Replaced Figure 11-3. • Replaced Figure 11-14.
May 2008	8.0.0	• Updated Figure 11–5. • Updated "Types of Power Analyses" on page 11–5. • Updated "Operating Conditions" on page 11–9. • Updated "PowerPlay Power Analyzer Compilation Report" on page 11–31. • Updated "Current Drawn from Voltage Supplies" on page 11–32.

5 System Debugging Tools Overview

5.1 About Intel FPGA System Debugging Tools

The Intel FPGA system debugging tools help you verify your FPGA designs. As your product requirements continue to increase in complexity, the time you spend on design verification continues to rise. This chapter provides a quick overview of the tools available in the system debugging suite and discusses the criteria for selecting the best tool for your design.

5.2 System Debugging Tools Portfolio

The Quartus Prime software provides a portfolio of system design debugging tools for real-time verification of your design. Each tool in the system debugging portfolio uses a combination of available memory, logic, and routing resources to assist in the debugging process.

The tools provide visibility by routing (or "tapping") signals in your design to debugging logic. The debugging logic is then compiled with your design and downloaded into the FPGA or CPLD for analysis. Because different designs can have different constraints and requirements, such as the number of spare pins available or the amount of logic or memory resources remaining in the physical device, you can choose a tool from the available debugging tools that matches the specific requirements for your design.

5.2.1 System Debugging Tools Comparison

Table 29. Debugging Tools Portfolio

Tool	Description	Typical Usage
System Console	Uses a Tcl interpreter to communicate with hardware modules instantiated in your design. You can use it with the Transceiver Toolkit to monitor or debug your design. System Console provides real-time in-system debugging capabilities. Using System Console, you can read from and write to Memory Mapped components in our system without a processor or additional software. System Console uses Tcl as the fundamental infrastructure, so you can source scripts, set variables, write procedures, and take advantage of all the features of the Tcl scripting language.	You need to perform system-level debugging. For example, if you have an Avalon®-MM slave or Avalon-ST interfaces, you can debug your design at a transaction level. The tool supports JTAG connectivity and TCP/IP connectivity to the target FPGA you wish to debug.
Transceiver Toolkit	The Transceiver Toolkit allows you to test and tune transceiver link signal quality through a combination of metrics. Auto Sweeping of physical medium attachment (PMA) settings allows you to quickly find an optimal solution.	You need to debug or optimize signal integrity of your board layout even before the actual design to be run on the FPGA is ready.

continued...

Tool	Description	Typical Usage
Signal Tap Logic Analyzer	This logic analyzer uses FPGA resources to sample test nodes and outputs the information to the Quartus Prime software for display and analysis.	You have spare on-chip memory and you want functional verification of your design running in hardware.
Signal Probe	This tool incrementally routes internal signals to I/O pins while preserving results from your last place-and-routed design.	You have spare I/O pins and you would like to check the operation of a small set of control pins using either an external logic analyzer or an oscilloscope.
Logic Analyzer Interface (LAI)	This tool multiplexes a larger set of signals to a smaller number of spare I/O pins. LAI allows you to select which signals are switched onto the I/O pins over a JTAG connection.	You have limited on-chip memory, and have a large set of internal data buses that you would like to verify using an external logic analyzer. Logic analyzer vendors, such as Tektronics and Agilent, provide integration with the tool to improve the usability of the tool.
In-System Sources and Probes	This utility provides an easy way to drive and sample logic values to and from internal nodes using the JTAG interface.	You want to prototype a front panel with virtual buttons for your FPGA design.
In-System Memory Content Editor	This tool displays and allows you to edit on-chip memory.	You would like to view and edit the contents of on-chip memory that is not connected to a Nios II processor. You can also use the tool when you do not want to have a Nios II debug core in your system.
Virtual JTAG Interface	This megafunction allows you to communicate with the JTAG interface so that you can develop your own custom applications.	You have custom signals in your design that you want to be able to communicate with.

Related Links

AN 799: Quick Debugging of Intel Arria 10 Designs Using Signal Probe and Rapid Recompile

5.2.2 System-Level Debugging Infrastructure

Intel FPGA on-chip debugging tools use the JTAG port to control and read-back data from debugging logic and signals under test.
When your design includes multiple debugging blocks, all of the on-chip debugging tools share the JTAG resource.

Note: For System Console, you explicitly insert debug IP cores into your design to enable debugging.

During compilation, the Quartus Prime software identifies the debugging blocks that use a JTAG interface and groups them under the JTAG Hub. This architecture allows you to instantiate multiple debugging tools in your design and use them simultaneously. The JTAG Hub appears in the design hierarchy of your project as a partition named `auto_fab_<number>`.

Related Links

Debug Infrastructure on page 320
During synthesis, the Compiler generates centralized debug managers—or hubs—for each region (static and PR) that contains debug agents. Each hub handles the debug agents in its partition.

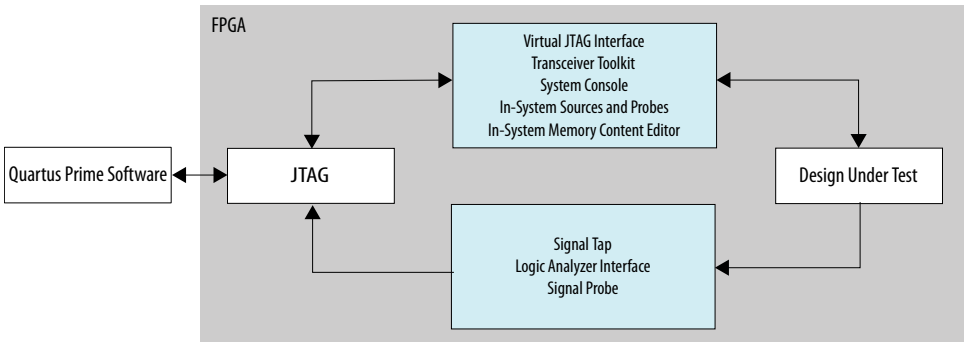
5.2.3 Debugging Ecosystem

The Quartus Prime software allows you to use the debugging tools in tandem to exercise and analyze the logic under test, and maximize closure. All debugging tools enable you to read back information gathered from the design nodes connected to the debugging logic.

Out of the set of debugging tools, the Signal Tap Logic Analyzer, the Logic Analyzer Interface, and the Signal Probe feature are general purpose troubleshooting tools optimized for probing signals in your register transfer level (RTL) netlist. In-System Sources and Probes, the Virtual JTAG Interface, System Console, Transceiver Toolkit, and In-System Memory Content Editor, allow you to read back data from defined breakpoints, and to input values into your design during runtime.

Taken together, the set of on-chip debugging tools form a debugging ecosystem. The set of tools can generate a stimulus to and solicit a response from the logic under test, providing a complete solution.

Figure 111. Debugging Ecosystem at Runtime



5.2.4 Debugging a Partial Reconfiguration Design with System Level Design Tools

Use the Quartus Prime software on-chip debugging tools, such as Signal Tap Logic Analyzer, In-System Sources and Probes Editor (IISP), In-System Memory Content Editor (ISMCE), or JTAG Avalon Master Bridge (JAMB) to verify your partial reconfiguration design.

Note: Only Signal Tap Logic Analyzer allows debugging of both the static and PR regions. Other tools support debugging only in the static region.

Related Links

Debugging Partial Reconfiguration Designs Using Signal Tap Logic Analyzer on page 319

You can debug your PR design using Signal Tap. Quartus Prime software provides debug capabilities that allow you to acquire signals in static and PR regions simultaneously.

5.2.5 About Analysis Tools for RTL Nodes

The Signal Tap Logic Analyzer, Signal Probe, and LAI are designed specifically for probing and debugging RTL signals at system speed. They are general-purpose analysis tools that enable you to tap and analyze any routable node from the FPGA or CPLD. If you have spare logic and memory resources, the Signal Tap Logic Analyzer is useful for providing fast functional verification of your design running on actual hardware.

Conversely, if logic and memory resources are tight and you require the large sample depths associated with external logic analyzers, both the LAI and the Signal Probe make it easy to view internal design signals using external equipment.

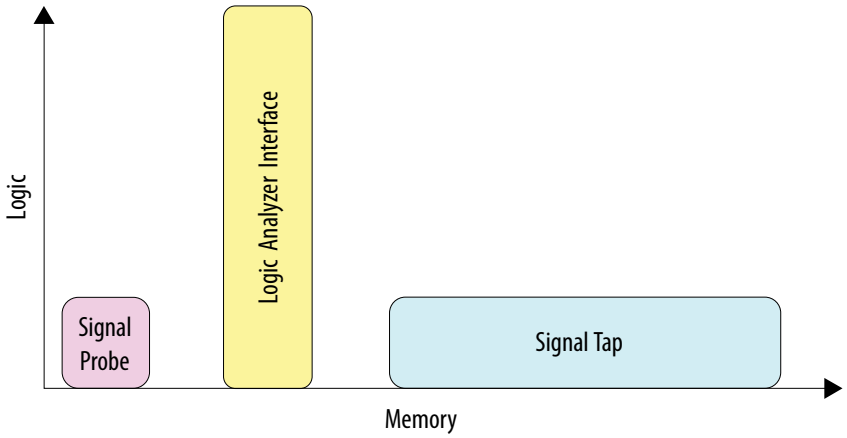
Note: The Signal Tap Logic Analyzer is not supported on CPLDs, because there are no memory resources available on these devices.

5.2.5.1 Resource Usage

The most important selection criteria for these three tools are the remaining resources on your device after implementing your design, and the number of spare pins.

Evaluate your debugging options early on in the design planning process to ensure that your board, your Quartus Prime project, and your design are set up to support the appropriate options. Planning early can reduce time spent during debugging and eliminate the necessary late changes to accommodate your selected methodologies.

Figure 112. Resource Usage per Debugging Tool



5.2.5.1.1 Overhead Logic

Any debugging tool that requires the use of a JTAG connection requires the SLD infrastructure logic, for communication with the JTAG interface and arbitration between any instantiated debugging modules. This overhead logic uses around 200 logic elements (LEs), a small fraction of the resources available in any of the supported devices. The overhead logic is shared between all available debugging modules in your design. Both the Signal Tap Logic Analyzer and the LAI use a JTAG connection.

For Signal Probe

Signal Probe requires very few on-chip resources. Because it requires no JTAG connection, Signal Probe uses no logic or memory resources. Signal Probe uses only routing resources to route an internal signal to a debugging test point.

For Logic Analyzer Interface

The LAI requires a small amount of logic to implement the multiplexing function between the signals under test, in addition to the SLD infrastructure logic. Because no data samples are stored on the chip, the LAI uses no memory resources.

For Signal Tap Logic Analyzer

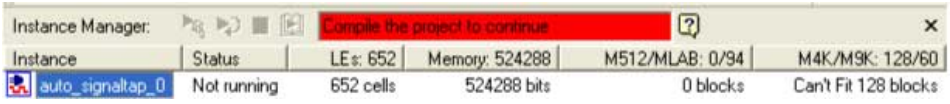
The Signal Tap Logic Analyzer requires both logic and memory resources. The number of logic resources used depends on the number of signals tapped and the complexity of the trigger logic. However, the amount of logic resources that the Signal Tap Logic Analyzer uses is typically a small percentage of most designs.

A baseline configuration consisting of the SLD arbitration logic and a single node with basic triggering logic contains approximately 300 to 400 Logic Elements (LEs). Each additional node you add to the baseline configuration adds about 11 LEs. Compared with logic resources, memory resources are a more important factor to consider for your design. Memory usage can be significant and depends on how you configure your Signal Tap Logic Analyzer instance to capture data and the sample depth that your design requires for debugging. For the Signal Tap Logic Analyzer, there is the added benefit of requiring no external equipment, as all of the triggering logic and storage is on the chip.

5.2.5.1.2 Resource Estimation

The resource estimation feature for the Signal Tap Logic Analyzer and the LAI allows you to quickly judge if enough on-chip resources are available before compiling the tool with your design.

Figure 113. Resource Estimator



5.2.5.2 Pin Usage

5.2.5.2.1 For Signal Probe

The ratio of the number of pins used to the number of signals tapped for the Signal Probe feature is one-to-one. Because this feature can consume free pins quickly, a typical application for this feature is routing control signals to spare pins for debugging.

5.2.5.2.2 For Logic Analyzer Interface

The ratio of the number of pins used to the number of signals tapped for the LAI is many-to-one. It can map up to 256 signals to each debugging pin, depending on available routing resources. The control of the active signals that are mapped to the spare I/O pins is performed via the JTAG port. The LAI is ideal for routing data buses to a set of test pins for analysis.

5.2.5.2.3 For Signal Tap Logic Analyzer

Other than the JTAG test pins, the Signal Tap Logic Analyzer uses no additional pins. All data is buffered using on-chip memory and communicated to the Signal Tap Logic Analyzer GUI via the JTAG test port.

5.2.5.3 Usability Enhancements

The Signal Tap Logic Analyzer, Signal Probe, and LAI tools can be added to your existing design with minimal effects. With the node finder, you can find signals to route to a debugging module without making any changes to your HDL files. Signal Probe inserts signals directly from your post-fit database. The Signal Tap Logic Analyzer and LAI support inserting signals from both pre-synthesis and post-fit netlists.

5.2.5.3.1 Incremental Routing

Signal Probe uses the incremental routing feature. The incremental routing feature runs only the Fitter stage of the compilation. This leaves your compiled design untouched, except for the newly routed node or nodes. With Signal Probe, you can save as much as 90% compile time of a full compilation.

5.2.5.3.2 Automation Via Scripting

As another productivity enhancement, all tools in the on-chip debugging tool set support scripting via the `quartus_stp` Tcl package. For the Signal Tap Logic Analyzer and the LAI, scripting enables user-defined automation for data collection while debugging in the lab. The System Console includes a full Tcl interpreter for scripting.

5.2.5.3.3 Remote Debugging

You can perform remote debugging of your system with the Quartus Prime software via the System Console. This feature allows you to debug equipment deployed in the field through an existing TCP/IP connection.

There are two Application Notes available to assist you.

- Application Note 624 describes how to set up your Nios II system to use the System Console to perform remote debugging.
- Application Note 693 describes how to set up your Intel FPGA SoC to use the SLD tools to perform remote debugging.

Related Links

- Application Note 624: Debugging with System Console over TCP/IP
- Application Note 693: Remote Debugging over TCP/IP for Intel FPGA SoC

5.2.6 Suggested On-Chip Debugging Tools for Common Debugging Features

Table 30. Tools for Common Debugging Features (1)

Feature	Signal Probe	Logic Analyzer Interface (LAI)	Signal Tap Logic Analyzer	Description
Large Sample Depth	N/A	X	—	An external logic analyzer used with the LAI has a bigger buffer to store more captured data than the Signal Tap Logic Analyzer. No data is captured or stored with Signal Probe.
Ease in Debugging Timing Issue	X	X	—	External equipment, such as oscilloscopes and mixed signal oscilloscopes (MSOs), can be used with either LAI or Signal Probe. When used with the LAI, external equipment provides you with access to timing mode, which allows you to debug combined streams of data.
Minimal Effect on Logic Design	X	X (2)	X (2)	The LAI adds minimal logic to a design, requiring fewer device resources. The Signal Tap Logic Analyzer has little effect on the design, because it is set as a separate design partition. Signal Probe incrementally routes nodes to pins, not affecting the design at all.
Short Compile and Recompile Time	X	X (2)	X (2)	Signal Probe attaches incrementally routed signals to previously reserved pins, requiring very little recompilation time to make changes to source signal selections. The Signal Tap Logic Analyzer and the LAI can refit their own design partitions to decrease recompilation time.
Triggering Capability	N/A	N/A	X	The Signal Tap Logic Analyzer offers triggering capabilities that are comparable to commercial logic analyzers.
I/O Usage	—	—	X	No additional output pins are required with the Signal Tap Logic Analyzer. Both the LAI and Signal Probe require I/O pin assignments.
Acquisition Speed	N/A	—	X	The Signal Tap Logic Analyzer can acquire data at speeds of over 200 MHz. The same acquisition speeds are obtainable with an external logic analyzer used with the LAI, but might be limited by signal integrity issues.
No JTAG Connection Required	X	—	X	A FPGA design with the LAI requires an active JTAG connection to a host running the Quartus Prime software. Signal Probe and Signal Tap do not require a host for debugging purposes.
No External Equipment Required	—	—	X	The Signal Tap Logic Analyzer logic is completely internal to the programmed FPGA device. No extra equipment is required other than a JTAG connection from a host running the Quartus Prime software or the stand-alone Signal Tap Logic Analyzer software. Signal Probe and the LAI require the use of external debugging equipment, such as multimeters, oscilloscopes, or logic analyzers.

Notes to Table:

1. • X indicates the recommended tools for the feature.
   • — indicates that while the tool is available for that feature, that tool might not give the best results.
   • N/A indicates that the feature is not applicable for the selected tool.

Related Links

AN 799: Quick Debugging of Intel Arria 10 Designs Using Signal Probe and Rapid Recompile

5.2.7 About Stimulus-Capable Tools

The In-System Memory Content Editor, In-System Sources and Probes, and Virtual JTAG interface enable you to use the JTAG interface as a general-purpose communication port.

Though all three tools can be used to achieve the same results, there are some considerations that make one tool easier to use in certain applications than others. In-System Sources and Probes is ideal for toggling control signals. The In-System Memory Content Editor is useful for inputting large sets of test data. Finally, the Virtual JTAG interface is well suited for more advanced users who want to develop their own customized JTAG solution.

System Console provides system-level debugging at a transaction level, such as with Avalon-MM slave or Avalon-ST interfaces. You can communicate to a chip through JTAG, and TCP/IP protocols. System Console uses a Tcl interpreter to communicate with hardware modules that you have instantiated into your design.

5.2.7.1 In-System Sources and Probes

In-System Sources and Probes is an easy way to access JTAG resources to both read and write to your design. You can start by instantiating a megafunction into your HDL code. The megafunction contains source ports and probe ports for driving values into and sampling values from the signals that are connected to the ports, respectively. Transaction details of the JTAG interface are abstracted away by the megafunction. During runtime, a GUI displays each source and probe port by instance and allows you to read from each probe port and drive to each source port. The GUI makes this tool ideal for toggling a set of control signals during the debugging process.

5.2.7.1.1 Push Button Functionality

A good application of In-System Sources and Probes is to use the GUI as a replacement for the push buttons and LEDs used during the development phase of a project. Furthermore, In-System Sources and Probes supports a set of scripting commands for reading and writing using `quartus_stp`. When used with the Tk toolkit, you can build your own graphical interfaces. This feature is ideal for building a virtual front panel during the prototyping phase of the design.

5.2.7.2 In-System Memory Content Editor

The In-System Memory Content Editor allows you to quickly view and modify memory content either through a GUI interface or through Tcl scripting commands. The In-System Memory Content Editor works by turning single-port RAM blocks into dual-port RAM blocks. One port is connected to your clock domain and data signals, and the other port is connected to the JTAG clock and data signals for editing or viewing.

5.2.7.2.1 Generate Test Vectors

Because you can modify a large set of data easily, a useful application for the In-System Memory Content Editor is to generate test vectors for your design. For example, you can instantiate a free memory block, connect the output ports to the logic under test (using the same clock as your logic under test on the system side),

and create the glue logic for the address generation and control of the memory. At runtime, you can modify the contents of the memory using either a script or the In-System Memory Content Editor GUI and perform a burst transaction of the data contents in the modified RAM block synchronous to the logic being tested.

5.2.7.3 Virtual JTAG Interface Megafunction

The Virtual JTAG Interface megafunction provides the finest level of granularity for manipulating the JTAG resource. This megafunction allows you to build your own JTAG scan chain by exposing all of the JTAG control signals and configuring your JTAG Instruction Registers (IRs) and JTAG Data Registers (DRs). During runtime, you control the IR/DR chain through a Tcl API, or with System Console. This feature is meant for users who have a thorough understanding of the JTAG interface and want precise control over the number and type of resources used.

5.2.7.4 System Console

System Console is a framework that you can launch from the Quartus Prime software to start services for performing various debugging tasks. System Console provides you with Tcl scripts and a GUI to access the Qsys Pro system integration tool to perform low-level hardware debugging of your design, as well as identify a module by its path, and open and close a connection to a Qsys Pro module. You can access your design at a system level for purposes of loading, unloading, and transferring designs to multiple devices. Also, System Console supports the Tk toolkit for building graphical interfaces.

5.2.7.4.1 Test Signal Integrity

System Console also allows you to access commands that allow you to control how you generate test patterns, as well as verify the accuracy of data generated by test patterns. You can use JTAG debug commands in System Console to verify the functionality and signal integrity of your JTAG chain. You can test clock and reset signals.

5.2.7.4.2 Board Bring-Up and Verification

You can use System Console to access programmable logic devices on your development board, perform board bring-up, and perform verification. You can also access software running on a Nios II or Intel FPGA SoC processor, as well as access modules that produce or consume a stream of bytes.

5.2.7.4.3 Test Link Signal Integrity with Transceiver Toolkit

Transceiver Toolkit runs from the System Console framework, and allows you to run automatic tests of your transceiver links for debugging and optimizing your transceiver designs. You can use the Transceiver Toolkit GUI to set up channel links in your transceiver devices, and then automatically run EyeQ and Auto Sweep testing to view a graphical representation of your test data.

5.3 Document Revision History

Table 31. Document Revision History

Date	Version	Changes
2017.05.08	17.0.0	• Combined Altera JTAG Interface and Required Arbitration Logic topics into a new updated topic named System-Level Debugging Infrastructure. • Added topic: Debug the Partial Reconfiguration Design with System Level Debugging Tools.
2016.10.31	16.1.0	• Implemented Intel rebranding.
2015.11.02	15.1.0	Changed instances of *Quartus II* to *Quartus Prime*.
June 2014	14.0.0	Added information that System Console supports the Tk toolkit.
November 2013	13.1.0	Dita conversion. Added link to Remote Debugging over TCP/IP for Altera SoC Application Note.
June 2012	12.0.0	Maintenance release.
November 2011	10.0.2	Maintenance release. Changed to new document template.
December 2010	10.0.1	Maintenance release. Changed to new document template.
July 2010	10.0.0	Initial release

Related Links

Altera Documentation Archive

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

6 Analyzing and Debugging Designs with System Console

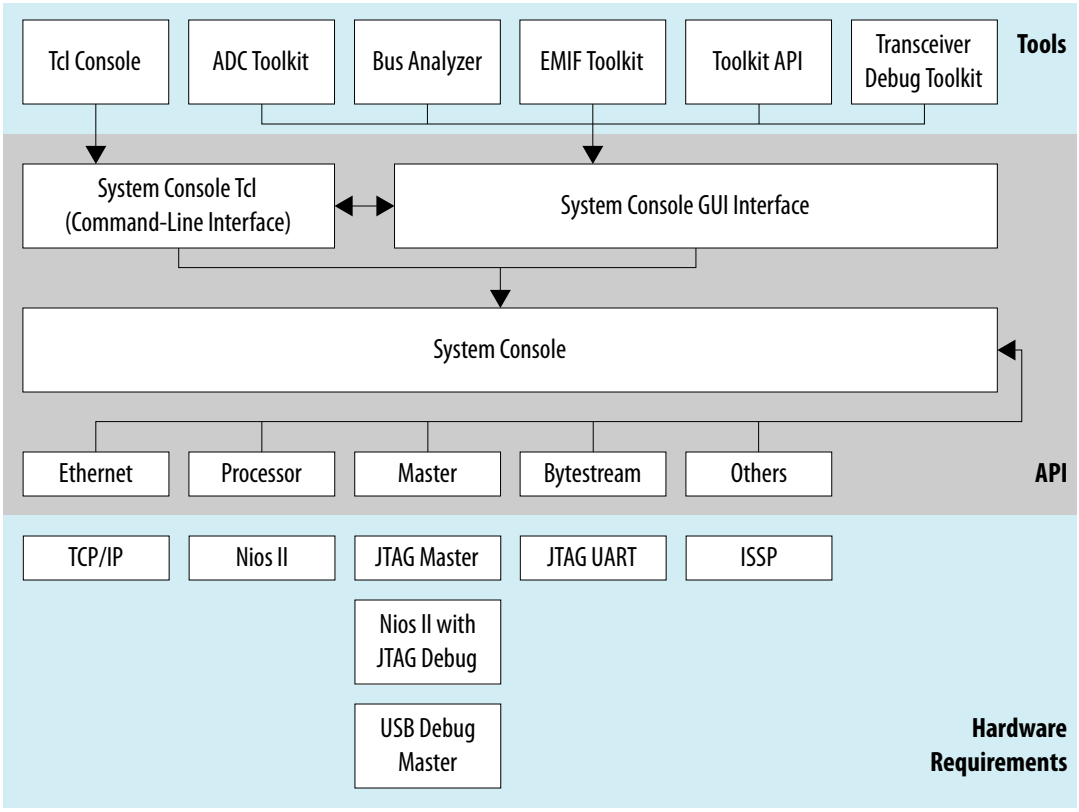
6.1 Introduction to System Console

System Console provides visibility into your design and allows you to perform system-level debug on a FPGA at run-time. System Console performs tests on debug-enabled Qsys Pro instantiated IP cores. A variety of debug services provide read and write access to elements in your design. You can perform the following tasks with System Console and the tools built on top of System Console:

- Bring up boards with both finalized and partially complete designs.
- Perform remote debug with internet access.
- Automate run-time verification through scripting across multiple devices in your system.
- Test serial links with point-and-click configuration tuning in the Transceiver Toolkit.
- Debug memory interfaces with the External Memory Interface Toolkit.
- Integrate your debug IP into the debug platform.
- Perform system verification with MATLAB/Simulink.

Figure 114. System Console Tools

(*Tools*) shows the applications that interact with System Console. The System Console API supports services that access your design in operation. Some services have specific hardware requirements.



Note: Use debug links to connect the host to the target you are debugging.

Related Links

- Introduction to Altera Memory Solution
  In *External Memory Interface Handbook Volume 1*
- Debugging Transceiver Links on page 227
  The Transceiver Toolkit helps you to optimize high-speed serial links in your board design.
- Application Note 693: Remote Hardware Debugging over TCP/IP for Intel SoC
- Application Note 624: Debugging with System Console over TCP/IP
- White Paper 01208: Hardware in the Loop from the MATLAB/Simulink Environment
- System Console Online Training

6.2 Debugging Flow with the System Console

To use the System Console you perform a series of steps:

1. Add an IP Core to your Qsys Pro system.
2. Generate your Qsys Pro system.
3. Compile your design.
4. Connect a board and program the FPGA.
5. Start System Console.
6. Locate and open a System Console service.
7. Perform debug operation(s) with the service.
8. Close the service.

6.3 IP Cores that Interact with System Console

System Console runs on your host computer and communicates with your running design through debug agents. Debug agents are soft-logic embedded in some IP cores that enable debug communication with the host computer.

You instantiate debug IP cores using the Qsys Pro IP Catalog. Some IP cores are enabled for debug by default, while you can enable debug for other IP cores through options in the parameter editor. Some debug agents have multiple purposes.

When you use IP cores with embedded debug in your design, you can make large portions of the design accessible. Debug agents allow you to read and write to memory and alter peripheral registers from the host computer.

Services associated with debug agents in the running design can open and close as needed. System Console determines the communication protocol with the debug agent. The communication protocol determines the best board connection to use for command and data transmission.

The Programmable SRAM Object File (`.sof`) provides the System Console with channel communication information. When System Console opens in the Quartus Prime software or Qsys Pro while your design is open, any existing `.sof` is automatically found and linked to the detected running device. In a complex system, you may need to link the design and device manually.

Related Links

WP-01170 System-Level Debugging and Monitoring of FPGA Designs

6.3.1 Services Provided through Debug Agents

By adding the appropriate debug agent to your design, System Console services can use the associated capabilities of the debug agent.

Table 32. Common Services for System Console

Service	Function	Debug Agent Providing Service
master	Access memory-mapped (Avalon-MM or AXI) slaves connected to the master interface.	• Nios II with debug • JTAG to Avalon Master Bridge • USB Debug Master
slave	Allows the host to access a single slave without needing to know the location of the slave in the host's memory map. Any slave that is accessible to a System Console master can provide this service.	• Nios II with debug • JTAG to Avalon Master Bridge • USB Debug Master If an SRAM Object File (`.sof`) is loaded, then slaves controlled by a debug master provide the slave service.
processor	• Start, stop, or step the processor. • Read and write processor registers.	Nios II with debug
JTAG UART	The JTAG UART is an Avalon-MM slave device that you can use in conjunction with System Console to send and receive byte streams.	JTAG UART

Note: The following IP cores in the IP Catalog do not support VHDL simulation generation in the current version of the Quartus Prime software:

- JTAG Debug Link
- JTAG Hub Controller System
- USB Debug Link

Related Links

- System Console Examples and Tutorials on page 214
  Intel provides examples for performing board bring-up, creating a simple dashboard, and programming a Nios II processor.
- System Console Commands on page 162

6.4 Starting System Console

6.4.1 Starting System Console from Nios II Command Shell

1. On the Windows Start menu, click **All Programs ➤ Intel ➤ Nios II EDS <*version*> ➤ Nios II<*version*> ➤ Command Shell.**.
2. Type `system-console`.
3. Type `-- help` for System Console help.
4. Type `system-console --project_dir=<project directory>` to point to a directory that contains `.qsf` or `.sof` files.

6.4.2 Starting Stand-Alone System Console

You can get the stand-alone version of System Console as part of the Quartus Prime software Programmer and Tools installer on the Altera website.

1. Navigate to the **Download Center** page and click the **Additional Software** tab.
2. On the Windows Start menu, click **All Programs ➤ Intel FPGA <*version*> ➤ Programmer and Tools ➤ System Console**.

Related Links

Intel Download Center

6.4.3 Starting System Console from Qsys Pro

Click **Tools ➤ System Console**.

6.4.4 Starting System Console from Quartus Prime

Click **Tools ➤ System Debugging Tools ➤ System Console**.

6.4.5 Customizing Startup

You can customize your System Console environment, as follows:

- Add commands to the `system_console_rc` configuration file located at:
  - `<$HOME>/system_console/system_console_rc.tcl`

    The file in this location is the user configuration file, which only affects the owner of the home directory.
- Specify your own design startup configuration file with the command-line argument `--rc_script=<path_to_script>`, when you launch System Console from the Nios II command shell.
- Use the `system_console_rc.tcl` file in combination with your custom `rc_script.tcl` file. In this case, the `system_console_rc.tcl` file performs System Console actions, and the `rc_script.tcl` file performs your debugging actions.

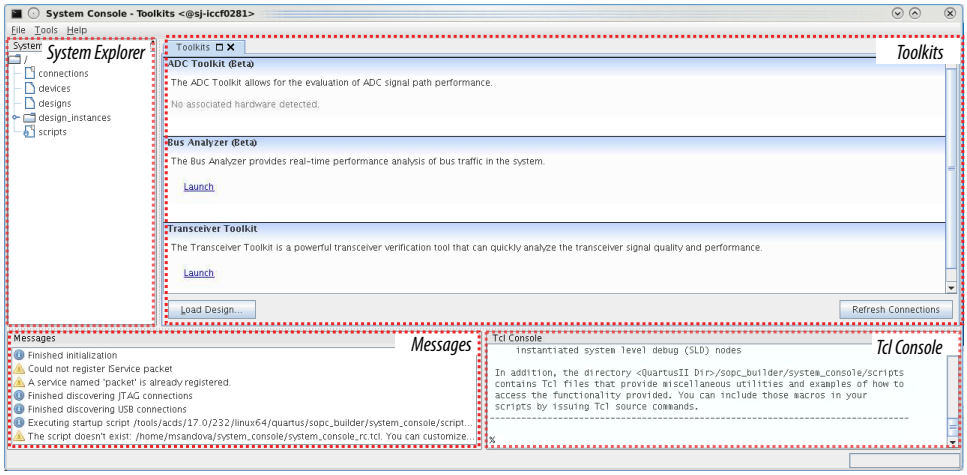
On startup, System Console automatically runs the Tcl commands in these files. The commands in the `system_console_rc.tcl` file run first, followed by the commands in the `rc_script.tcl` file.

6.5 System Console GUI

The System Console GUI consists of a main window with multiple panes, and allows you to interact with the design currently running on the host computer.

- **System Explorer**—Displays the hierarchy of the System Console virtual file system in your design, including board connections, devices, designs, and scripts.
- **Toolkits**—Displays available toolkits including the ADC Toolkit, Transceiver Toolkit, Toolkits, GDB Server Control Panel, and Bus Analyzer. Click the **Tools** menu to launch applications.
- **Tcl Console**—A window that allows you to interact with your design using Tcl scripts, for example, sourcing scripts, writing procedures, and using System Console API.
- **Messages**—Displays status, warning, and error messages related to connections and debug actions.

Figure 115. System Console GUI



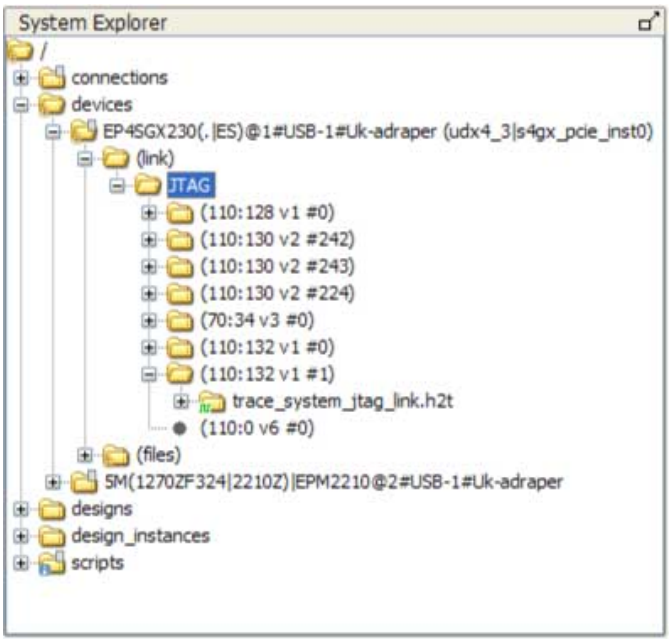
6.5.1 System Explorer Pane

The **System Explorer** pane displays the virtual file system for all connected debugging IP cores, and contains the following information:

- **Devices** folder—Contains information about each device connected to System Console.
- **Scripts** folder—Stores scripts for easy execution.
- **Connections** folder—Displays information about the board connections which are visible to System Console, such as USB Blaster. Multiple connections are possible.
- **Designs** folder—Displays information about Quartus Prime project designs connected to System Console. Each design represents a `.sof` file that is loaded.

The **Devices** contains a sub-folder for each device currently connected to System Console. Each device sub-folder contains a **(link)** folder and sometimes contains a **(files)** folder. The **(link)** folder shows debug agents (and other hardware) that System Console can access. The **(files)** folder contains information about the design files loaded from the Quartus Prime project for the device.

Figure 116. System Explorer Pane



- The figure above shows the **EP4SGX230** folder under the **Device** folder, which contains a **(link)** folder. The **(link)** folder contains a **JTAG** folder, which describes the active debug connections to this device, for example, JTAG, USB, Ethernet, and agents connected to the EP4SGX230 device via a JTAG connection.
- Folders with a context menu display a context menu icon. Right-click these folders to view the context menu. For example, the **Connections** folder above shows a context menu icon.
- Folders that have messages display a message icon. Mouse-over these folders to view the messages. For example, the **Scripts** folder above a message icon.
- Debug agents that sense the clock and reset state of the target show an information or error message with a clock status icon. The icon indicates whether the clock is running (information, green), stopped (error, red), or running but in reset (error, red). For example, the **trace_system_jtag_link.h2t** folder above has a running clock.

6.6 System Console Commands

The console commands enable testing. Use console commands to identify a service by its path, and to open and close the connection. The `path` that identifies a service is the first argument to most System Console commands.

To initiate a service connection, do the following:

1. Identify a service by specifying its path with the `get_service_paths` command.
2. Open a connection to the service with the `claim_service` command.
3. Use Tcl and System Console commands to test the connected device.
4. Close a connection to the service with the `close_service` command

Note: For all Tcl commands, the <*format*> argument must come first.

Table 33. System Console Commands

Command	Arguments	Function
`get_service_types`	N/A	Returns a list of service types that System Console manages. Examples of service types include master, bytestream, processor, sld, jtag_debug, device, and design.
`get_service_paths`	• <*service-type*> • <*device*>—Returns services in the same specified device. The argument can be a device or another service in the device. • <*hpath*>—Returns services whose `hpath` starts with the specified prefix. • <*type*>—Returns services whose debug type matches this value. Particularly useful when opening slave services. • <*type*>—Returns services on the same development boards as the argument. Specify a board service, or any other service on the same board.	Allows you to filter the services which are returned.
`claim_service`	• <*service-type*> • <*service-path*> • <*claim-group*> • <*claims*>	Provides finer control of the portion of a service you want to use. `claim_service` returns a new path which represents a use of that service. Each use is independent. Calling claim_service multiple times returns different values each time, but each allows access to the service until closed.
`close_service`	• <*service-type*> • <*service-path*>	Closes the specified service type at the specified path.
`is_service_open`	• <*service-type*> • <*service-type*>	Returns 1 if the service type provided by the path is open, 0 if the service type is closed.
`get_services_to_add`	N/A	Returns a list of all services that are instantiable with the `add_service` command.
`add_service`	• <*service-type*> • <*instance-name*> • *optional-parameters*	Adds a service of the specified service type with the given instance name. Run `get_services_to_add` to retrieve a list of instantiable services. This command returns the path where the service was added. Run `help add_service` <*service-type*> to get specific help about that service type, including any parameters that might be required for that service.
`add_service gdbserver`	• <*Processor Service*> • <*port number*>	Instantiates a gdbserver.

continued...

Command	Arguments	Function
`add_service tcp`	• *<instance name>* • *<ip_addr>* • *<port_number>*	Allows you to connect to a TCP/IP port that provides a debug link over ethernet. See AN693 (*Remote Hardware Debugging over TCP/IP for Intel FPGA SoC*) for more information.
`add_service transceiver_channel_rx`	• *<data_pattern_checker>* • *<path>* • *<transceiver path>* • *<transceiver channel address>* • *<reconfig path>* • *<reconfig channel address>*	Instantiates a Transceiver Toolkit receiver channel.
`add_service transceiver_channel_tx`	• *<data_pattern_generator>* • *<path>* • *<transceiver path>* • *<transceiver channel address>* • *<reconfig path>* • *<reconfig channel address>*	Instantiates a Transceiver Toolkit transmitter channel.
`add_service transceiver_debug_link`	• *<transceiver_channel_tx path>* • *<transceiver_channel_rx path>*	Instantiates a Transceiver Toolkit debug link.
`get_version`	N/A	Returns the current System Console version and build number.
`get_claimed_services`	• *<claim>*	For the given claim group, returns a list of services claimed. The returned list consists of pairs of paths and service types. Each pair is one claimed service.
`refresh_connections`	N/A	Scans for available hardware and updates the available service paths if there have been any changes.
`send_message`	• *<level>* • *<message>*	Sends a message of the given level to the message window. Available levels are info, warning, error, and debug.

Related Links

Remote Hardware Debugging over TCP/IP for SoC Devices

6.7 Running System Console in Command-Line Mode

You can run System Console in command line mode and either work interactively or run a Tcl script. System Console prints the output in the console window.

- `--cli`—Runs System Console in command-line mode.
- `--project_dir=`*`<project dir>`*—Directs System Console to the location of your hardware project. Also works in GUI mode.
- `--script=`*`<your script>`*`.tcl`—Directs System Console to run your Tcl script.
- `--help`— Lists all available commands. Typing `--help` *<command name>* provides the syntax and arguments of the command.

System Console provides command completion if you type the beginning letters of a command and then press the **Tab** key.

6.8 System Console Services

Intel's System Console services provide access to hardware modules instantiated in your FPGA. Services vary in the type of debug access they provide.

6.8.1 Locating Available Services

System Console uses a virtual file system to organize the available services, which is similar to the `/dev location` on Linux systems. Board connection, device type, and IP names are all part of a service path. Instances of services are referred to by their unique service path in the file system. To retrieve service paths for a particular service, use the command `get_service_paths` <*service-type*>.

Example 8. Locating a Service Path

```
#We are interested in master services.
set service_type "master"

#Get all the paths as a list.
set master_service_paths [get_service_paths $service_type]

#We are interested in the first service in the list.
set master_index 0

#The path of the first master.
set master_path [lindex $master_service_paths $master_index]

#Or condense the above statements into one statement:
set master_path [lindex [get_service_paths master] 0]
```

System Console commands require service paths to identify the service instance you want to access. The paths for different components can change between runs of System Console and between versions. Use the `get_service_paths` command to obtain service paths.

The string values of service paths change with different releases of the tool. Use the `marker_node_info` command to get information from the path.

System Console automatically discovers most services at startup. System Console automatically scans for all JTAG and USB-based service instances and retrieves their service paths. System Console does not automatically discover some services, such as TCP/IP. Use `add_service` to inform System Console about those services.

Example 9. Marker_node_info

Use the `marker_node_info` command to get information about the discovered services.

```
set slave_path [get_service_paths -type altera_avalon_uart.slave slave]
array set uart_info [marker_node_info $slave_path]
echo $uart_info(full_hpath)
```

6.8.2 Opening and Closing Services

After you have a service path to a particular service instance, you can access the service for use.

The `claim_service` command directs System Console to start using a particular service instance, and with no additional arguments, claims a service instance for exclusive use.

Example 10. Opening a Service

```
set service_type "master"
set claim_path [claim_service $service_type $master_path mylib];#Claims service.
```

You can pass additional arguments to the `claim_service` command to direct System Console to start accessing a particular portion of a service instance. For example, if you use the master service to access memory, then use `claim_service` to only access the address space between `0x0` and `0x1000`. System Console then allows other users to access other memory ranges, and denies access to the claimed memory range. The `claim_service` command returns a newly created service path that you can use to access your claimed resources.

You can access a service after you open it. When you finish accessing a service instance, use the `close_service` command to direct System Console to make this resource available to other users.

Example 11. Closing a Service

```
close_service master $claim_path; #Closes the service.
```

6.8.3 SLD Service

The SLD Service shifts values into the instruction and data registers of SLD nodes and captures the previous value. When interacting with a SLD node, start by acquiring exclusive access to the node on an opened service.

Example 12. SLD Service

```
set timeout_in_ms 1000
set lock_failed [sld_lock $sld_service_path $timeout_in_ms]
```

This code attempts to lock the selected SLD node. If it is already locked, `sld_lock` waits for the specified timeout. Confirm the procedure returns non-zero before proceeding. Set the instruction register and capture the previous one:

```
if {$lock_failed} {
    return
}
set instr 7
set delay_us 1000
set capture [sld_access_ir $sld_service_path $instr $delay_us]
```

The 1000 microsecond delay guarantees that the following SLD command executes least 1000 microseconds later. Data register access works the same way.

```
set data_bit_length 32
set delay_us 1000
set data_bytes [list 0xEF 0xBE 0xAD 0xDE]
set capture [sld_access_dr $sld_service_path $data_bit_length $delay_us \
$data_bytes]
```

Shift count is specified in bits, but the data content is specified as a list of bytes. The capture return value is also a list of bytes. Always unlock the SLD node once finished with the SLD service.

```
sld_unlock $sld_service_path
```

Related Links

Virtual JTAG IP Core User Guide

6.8.3.1 SLD Commands

Table 34. SLD Commands

Command	Arguments	Function
`sld_access_ir`	*<claim-path>* *<ir-value>* *<delay>* (in µs)	Shifts the instruction value into the instruction register of the specified node. Returns the previous value of the instruction. If the *<delay>* parameter is non-zero, then the JTAG clock is paused for this length of time after the access.
`sld_access_dr`	*<service-path>* *<size_in_bits>* *<delay-in-µs>*, *<list_of_byte_values>*	Shifts the byte values into the data register of the SLD node up to the size in bits specified. If the *<delay>* parameter is non-zero, then the JTAG clock is paused for at least this length of time after the access. Returns the previous contents of the data register.
`sld_lock`	*<service-path>* *<timeout-in-milliseconds>*	Locks the SLD chain to guarantee exclusive access. Returns 0 if successful. If the SLD chain is already locked by another user, tries for *<timeout>*ms before throwing a Tcl error. You can use the catch command if you want to handle the error.
`sld_unlock`	*<service-path>*	Unlocks the SLD chain.

6.8.4 In-System Sources and Probes Service

The In-System Sources and Probes (ISSP) service provides scriptable access to the `altsource_probe` IP core in a similar manner to using the **In-System Sources and Probes Editor** in the Quartus Prime software.

Example 13. ISSP Service

Before you use the ISSP service, ensure your design works in the **In-System Sources and Probes Editor**. In System Console, open the service for an ISSP instance.

```
set issp_index 0
set issp [lindex [get_service_paths issp] 0]
set claimed_issp [claim_service issp $issp mylib]
```

View information about this particular ISSP instance.

```
array set instance_info [issp_get_instance_info $claimed_issp]
set source_width $instance_info(source_width)
set probe_width $instance_info(probe_width)
```

The Quartus Prime software reads probe data as a single bitstring of length equal to the probe width.

```
set all_probe_data [issp_read_probe_data $claimed_issp]
```

As an example, you can define the following procedure to extract an individual probe line's data.

```
proc get_probe_line_data {all_probe_data index} {
    set line_data [expr { ($all_probe_data >> $index) & 1 }]
    return $line_data
}
set initial_all_probe_data [issp_read_probe_data $claim_issp]
set initial_line_0 [get_probe_line_data $initial_all_probe_data 0]
set initial_line_5 [get_probe_line_data $initial_all_probe_data 5]
# ...
set final_all_probe_data [issp_read_probe_data $claimed_issp]
set final_line_0 [get_probe_line_data $final_all_probe_data 0]
```

Similarly, the Quartus Prime software writes source data as a single bitstring of length equal to the source width.

```
set source_data 0xDEADBEEF
issp_write_source_data $claimed_issp $source_data
```

The currently set source data can also be retrieved.

```
set current_source_data [issp_read_source_data $claimed_issp]
```

As an example, you can invert the data for a 32-bit wide source by doing the following:

```
set current_source_data [issp_read_source_data $claimed_issp]
set inverted_source_data [expr { $current_source_data ^ 0xFFFFFFFF }]
issp_write_source_data $claimed_issp $inverted_source_data
```

6.8.4.1 In-System Sources and Probes Commands

Note: The valid values for ISSP claims include `read_only`, `normal`, and `exclusive`.

Table 35. In-System Sources and Probes Commands

Command	Arguments	Function

`issp_get_instance_info`	*<service-path>*	Returns a list of the configurations of the In-System Sources and Probes instance, including:
		`instance_index`
		`instance_name`
		`source_width`

continued...

Command	Arguments	Function
		probe_width
issp_read_probe_data	<*service-path*>	Retrieves the current value of the probe input. A hex string is returned representing the probe port value.
issp_read_source_data	<*service-path*>	Retrieves the current value of the source output port. A hex string is returned representing the source port value.
issp_write_source_data	<*service-path*> <*source-value*>	Sets values for the source output port. The value can be either a hex string or a decimal value supported by the System Console Tcl interpreter.

6.8.5 Monitor Service

The monitor service builds on top of the master service to allow reads of Avalon-MM slaves at a regular interval. The service is fully software-based. The monitor service requires no extra soft-logic. This service streamlines the logic to do interval reads, and it offers better performance than exercising the master service manually for the reads.

Example 14. Monitor Service

Start by determining a master and a memory address range that you are interested in polling continuously.

```
set master_index     0
set master [lindex [get_service_paths master] $master_index]
set address          0x2000
set bytes_to_read    100
set read_interval_ms 100
```

You can use the first master to read 100 bytes starting at address 0x2000 every 100 milliseconds. Open the monitor service:

```
set monitor [lindex [get_service_paths monitor] 0]
set claimed_monitor [claim_service monitor $monitor mylib]
```

Notice that the master service was not opened. The monitor service opens the master service automatically. Register the previously-defined address range and time interval with the monitor service:

```
monitor_add_range $claimed_monitor $master $address $bytes_to_read
monitor_set_interval $claimed_monitor $read_interval_ms
```

You can add more ranges. You must define the result at each interval:

```
global monitor_data_buffer
set monitor_data_buffer [list]
proc store_data {monitor master address bytes_to_read} {
  global monitor_data_buffer
  set data [monitor_read_data $claimed_monitor $master $address $bytes_to_read]
  lappend monitor_data_buffer $data
}
```

The code example above, gathers the data and appends it with a global variable. `monitor_read_data` returns the range of data polled from the running design as a list. In this example, data will be a 100-element list. This list is then appended as a

single element in the `monitor_data_buffer` global list. If this procedure takes longer than the interval period, the monitor service may have to skip the next one or more calls to the procedure. In this case, `monitor_read_data` will return the latest data polled. Register this callback with the opened monitor service:

```
set callback [list store_data $claimed_monitor $master $address $bytes_to_read]
monitor_set_callback $claimed_monitor $callback
```

Use the callback variable to call when the monitor finishes an interval. Start monitoring:

```
monitor_set_enabled $claimed_monitor 1
```

Immediately, the monitor reads the specified ranges from the device and invokes the callback at the specified interval. Check the contents of `monitor_data_buffer` to verify this. To turn off the monitor, use 0 instead of 1 in the above command.

6.8.5.1 Monitor Commands

You can use the Monitor commands to read many Avalon-MM slave memory locations at a regular interval.

Under normal load, the monitor service reads the data after each interval and then calls the callback. If the value you read is timing sensitive, you can use the `monitor_get_read_interval` command to read the exact time between the intervals at which the data was read.

Under heavy load, or with a callback that takes a long time to execute, the monitor service skips some callbacks. If the registers you read do not have side effects (for example, they read the total number of events since reset), skipping callbacks has no effect on your code. The `monitor_read_data` command and `monitor_get_read_interval` command are adequate for this scenario.

If the registers you read have side effects (for example, they return the number of events since the last read), you must have access to the data that was read, but for which the callback was skipped. The `monitor_read_all_data` and `monitor_get_all_read_intervals` commands provide access to this data.

Table 36. Main Monitoring Commands

Command	Arguments	Function
`monitor_add_range`	*<service-path>* *<target-path>* *<address>* *<size>*	Adds a contiguous memory address into the monitored memory list. *<service path>* is the value returned when you opened the service. *<target-path>* argument is the name of a master service to read. The address is within the address space of this service. *<target-path>* is returned from `[lindex [get_service_paths master] n]` where *n* is the number of the master service. *<address>* and *<size>* are relative to the master service.
`monitor_set_callback`	*<service-path>* *<Tcl-expression>*	Defines a Tcl expression in a single string that will be evaluated after all the memories monitored by this service are read. Typically, this expression should be specified as a Tcl procedure call with necessary argument passed in.

continued...

Command	Arguments	Function
monitor_set_interval	<service-path> <interval>	Specifies the frequency of the polling action by specifying the interval between two memory reads. The actual polling frequency varies depending on the system activity. The monitor service will try to keep it as close to this specification as possible.
monitor_get_interval	<service-path>	Returns the current interval set which specifies the frequency of the polling action.
monitor_set_enabled	<service-path> <enable(1)/ disable(0)>	Enables and disables monitoring. Memory read starts after this is enabled, and Tcl callback is evaluated after data is read.

Table 37. Monitor Callback Commands

Command	Arguments	Function
monitor_add_range	<service-path> <target-path> <address> <size>	Adds contiguous memory addresses into the monitored memory list. The <target-path> argument is the name of a master service to read. The address is within the address space of this service.
monitor_set_callback	<service-path> <Tcl-expression>	Defines a Tcl expression in a single string that will be evaluated after all the memories monitored by this service are read. Typically, this expression should be specified as a Tcl procedure call with necessary argument passed in.
monitor_read_data	<service-path> <target-path> <address> <size>	Returns a list of 8-bit values read from the most recent values read from device. The memory range specified must be the same as the monitored memory range as defined by monitor_add_range.
monitor_read_all_data	<service-path> <target-path> <address> <size>	Returns a list of 8-bit values read from all recent values read from device since last Tcl callback. The memory range specified must be within the monitored memory range as defined by monitor_add_range.
monitor_get_read_interval	<service-path> <target-path> <address> <size>	Returns the number of milliseconds between last two data reads returned by monitor_read_data.
monitor_get_all_read_intervals	<service-path> <target-path> <address> <size>	Returns a list of intervals in milliseconds between two reads within the data returned by monitor_read_all_data.
monitor_get_missing_event_count	<service-path>	Returns the number of callback events missed during the evaluation of last Tcl callback expression.

6.8.6 Device Service

The device service supports device-level actions.

Example 15. Programming

You can use the device service with Tcl scripting to perform device programming.

```
set device_index 0 ; #Device index for target
set device [lindex [get_service_paths device] $device_index]
set sof_path [file join project_path output_files project_name.sof]
device_download_sof $device $sof_path
```

To program, all you need are the device service path and the file system path to a `.sof`. Ensure that no other service (e.g. master service) is open on the target device or else the command fails. Afterwards, you may do the following to check that the design linked to the device is the same one programmed:

```
device_get_design $device
```

6.8.6.1 Device Commands

The device commands provide access to programmable logic devices on your board. Before you use these commands, identify the path to the programmable logic device on your board using the `get_service_paths`.

Table 38. Device Commands

Command	Arguments	Function
`device_download_sof`	*<service_path>* *<sof-file-path>*	Loads the specified `.sof` to the device specified by the path.
`device_get_connections`	*<service_path>*	Returns all connections which go to the device at the specified path.
`device_get_design`	*<device_path>*	Returns the design this device is currently linked to.

6.8.7 Design Service

You can use design service commands to work with Quartus Prime design information.

Example 16. Load

When you open System Console from the Quartus Prime software or Qsys Pro, the current project's debug information is sourced automatically if the `.sof` has been built. In other situations, you can load manually.

```
set sof_path [file join project_dir output_files project_name.sof]
set design [design_load $sof_path]
```

System Console is now aware that this particular `.sof` has been loaded.

Example 17. Linking

Once a `.sof` is loaded, System Console automatically links design information to the connected device. The resultant link persists and you can choose to unlink or reuse the link on an equivalent device with the same `.sof`.

You can perform manual linking.

```
set device_index 0; # Device index for our target
set device [lindex [get_service_paths device] $device_index]
design_link $design $device
```

Manually linking fails if the target device does not match the design service.

Linking fails even if the `.sof` programmed to the target is not the same as the design `.sof`.

6.8.7.1 Design Service Commands

Design service commands load and work with your design at a system level.

Table 39. Design Service Commands

Command	Arguments	Function
`design_load`	*<quartus-project-path>*, *<sof-file-path>*, or *<qpf-file-path>*	Loads a model of a Quartus Prime design into System Console. Returns the design path. For example, if your Quartus Prime Project File (`.qpf`) is in `c:/projects/loopback`, type the following command: `design_load {c:\projects\loopback\}`
`design_link`	*<design-path>* *<device-service-path>*	Links a Quartus Prime logical design with a physical device. For example, you can link a Quartus Prime design called **2c35_quartus_design** to a 2c35 device. After you create this link, System Console creates the appropriate correspondences between the logical and physical submodules of the Quartus Prime project.
`design_extract_debug_files`	*<design-path>* *<zip-file-name>*	Extracts debug files from a `.sof` to a zip file which can be emailed to *Intel FPGA Support* for analysis. You can specify a design path of {} to unlink a device and to disable auto linking for that device.
`design_get_warnings`	*<design-path>*	Gets the list of warnings for this design. If the design loads correctly, then an empty list returns.

6.8.8 Bytestream Service

The bytestream service provides access to modules that produce or consume a stream of bytes. Use the bytestream service to communicate directly to the IP core that provides bytestream interfaces, such as the Altera JTAG UART or the Avalon-ST JTAG interface.

Example 18. Bytestream Service

The following code finds the bytestream service for your interface and opens it.

```
set bytestream_index 0
set bytestream [lindex [get_service_paths bytestream] $bytestream_index]
set claimed_bytestream [claim_service bytestream $bytestream mylib]
```

To specify the outgoing data as a list of bytes and send it through the opened service:

```
set payload [list 1 2 3 4 5 6 7 8]
bytestream_send $claimed_bytestream $payload
```

Incoming data also comes as a list of bytes.

```
set incoming_data [list]
while {[llength $incoming_data] ==0} {
    set incoming_data [bytestream_receive $claimed_bytestream 8]
}
```

Close the service when done.

```
close_service bytestream $claimed_bytestream
```

6.8.8.1 Bytestream Commands

Table 40. Bytestream Commands

Command	Arguments	Function
`bytestream_send`	*<service-path>* *<values>*	Sends the list of bytes to the specified bytestream service. Values argument is the list of bytes to send.
`bytestream_receive`	*<service-path>* *<length>*	Returns a list of bytes currently available in the specified services receive queue, up to the specified limit. Length argument is the maximum number of bytes to receive.

6.8.9 JTAG Debug Service

The JTAG Debug service allows you to check the state of clocks and resets within your design.

The following is a JTAG Debug design flow example.

1. To identify available JTAG Debug paths:

```
get_service_paths jtag_debug
```

2. To select a JTAG Debug path:

```
set jtag_debug_path [lindex [get_service_paths jtag_debug] 0]
```

3. To claim a JTAG Debug service path:

```
 set claim_jtag_path [claim_service jtag_debug$jtag_debug_path mylib]
```

4. Running the JTAG Debug service:

```
jtag_debug_reset_system $claim_jtag_path
jtag_debug_loop $claim_jtag_path [list 1 2 3 4 5]
```

6.8.9.1 JTAG Debug Commands

JTAG Debug commands help debug the JTAG Chain connected to a device.

Table 41. JTAG Debug Commands

Command	Argument	Function
`jtag_debug_loop`	*<service-path>* *<list_of_byte_values>*	Loops the specified list of bytes through a loopback of `tdi` and `tdo` of a system-level debug (SLD) node. Returns the list of byte values in the order that they were received. Blocks until all bytes are received. Byte values have the `0x` (hexadecimal) prefix and are delineated by spaces.
`jtag_debug_sample_clock`	*<service-path>*	Returns the value of the clock signal of the system clock that drives the module's system interface. The clock value is sampled asynchronously; consequently, you may need to sample the clock several times to guarantee that it is toggling.

continued...

Command	Argument	Function
`jtag_debug_sample_reset`	*<service-path>*	Returns the value of the `reset_n` signal of the Avalon-ST JTAG Interface core. If `reset_n` is low (asserted), the value is 0 and if `reset_n` is high (deasserted), the value is 1.
`jtag_debug_sense_clock`	*<service-path>*	Returns the result of a sticky bit that monitors for system clock activity. If the clock has toggled since the last execution of this command, the bit is 1. Returns `true` if the bit has ever toggled and otherwise returns `false`. The sticky bit is reset to 0 on read.
`jtag_debug_reset_system`	*<service-path>*	Issues a reset request to the specified service. Connectivity within your device determines which part of the system is reset.

6.9 Working with Toolkits

The Toolkit API allows you to create custom tools to visualize and interact with your design debug data. The Toolkit API provides graphical widgets in the form of buttons and text fields, which can leverage user input to interact with debug logic. You can use Toolkit API with the Quartus Prime software versions 14.1 and later. The Toolkit API is the successor to the Dashboard service.

Toolkits you create with the Toolkit API require the following files:

- XML file that describes the toolkit (`.toolkit` file).
- Tcl file that implements the toolkit GUI.

6.9.1 Convert your Dashboard Scripts to Toolkit API

Convert your Dashboard scripts to work with the Toolkit API by following these steps:

1. Create a `.toolkit` file.
2. Modify your dashboard script:
   a. Remove the `add_service dashboard <name of service>` command.
   b. Change `dashboard_<command>` to `toolkit_<command>`.
   c. Change `open_service` to `claim_service`

   For example:

```
open_service slave $path
master_read_memory $path address count
```

   becomes

```
set c [claim_service slave $path lib {}]
master_read_memory $c address count
```

6.9.2 Creating a Toolkit Description File

A toolkit description file (`.toolkit`) is a XML file which provides the registration data for a toolkit.

Include the following attributes in your toolkit description file:

Table 42. Attributes in Toolkit Description File

Attribute name	Purpose
`name`	Internal toolkit file name.
`displayName`	Toolkit display name to appear in the GUI.
`addMenuItem`	Whether the System Console **Tools ➤ Toolkits** menu displays the toolkit.

Table 43. Toolkit child elements

Attribute name	Purpose
`description`	Description of the purpose of the toolkit.
`file`	Path to `.tcl` file containing the toolkit implementation.
`icon`	Path to icon to display as the toolkit launcher button in System Console *Note:* The `.png` 64x64 format is preferred. If the icon does not take up the whole space, ensure that the background is transparent.
`requirement`	If the toolkit works with a particular type of hardware, this attribute specifies the debug type name of the hardware. This attribute enables automatic discovery of the toolkit. The syntax of a toolkit's debug type name is: • Name of the `hw.tcl` component. • dot. • Name of the interface within that component which the toolkit uses. For example: `<hw.tcl name>.<interface name>`.

Example 19. .toolkit Description File

```
<?xml version="1.0" encoding="UTF-8"?>
        <toolkit name="toolkit_example" displayName="Toolkit Example"
addMenuItem="true">
        <file> toolkit_example.tcl </file>
        </toolkit>
```

Related Links

Matching Toolkits with IP Cores on page 177
You can match your toolkit with any IP core:

6.9.3 Registering a Toolkit

Use the `toolkit_register` command in the System Console to make your toolkit available. Remember to specify the path to the `.toolkit` file. Registering a toolkit does not create an instance of the toolkit GUI.

```
toolkit_register <toolkit_file>
```

6.9.4 Launching a Toolkit

With the System Console, you can launch pre-registered toolkits in a number of ways:

- Click **Tools ➤ Toolkits**.
- Use the **Toolkits** tab. Each toolkit has a description, a detected hardware list, and a launch button.
- Use following command:

```
toolkit_open <.toolkit_file_name>
```

You can launch a toolkit in the context of a hardware resource associated with a toolkit type. If you use the command:

```
toolkit_open <toolkit_name> <context>
```

the toolkit Tcl can retrieve the context by typing

```
set context [toolkit_get_context]
```

Related Links

toolkit_get_context on page 188

6.9.5 Matching Toolkits with IP Cores

You can match your toolkit with any IP core:

- When searching for IP, the toolkit looks for debug markers and matches IP cores to the toolkit requirements. In the toolkit file, use the requirement attribute to specify a debug type, as follows:

```
<requirement><type>debug.type-name</type></requirement
```

- Create debug assignments in the `hw.tcl` for an IP core. `hw.tcl` files are available when you load the design in System Console.
- System Console discovers debug markers from identifiers in the hardware and associates with IP, without direct knowledge of the design.

6.9.6 Toolkit API

The Toolkit API service enables you to construct GUIs for visualizing and interacting with debug data. The Toolkit API is a graphical pane for the layout of your graphical widgets, which include buttons and text fields. Widgets pull data from other System Console services. Similarly, widgets use services to leverage user input to act on debug logic in your design.

Properties

Widget properties can push and pull information to the user interface. Widgets have properties specific to their type. For example, when you click a button, the button property `onClick` performs an action. A label widget does not have the same property, because the widget does not perform an action on click operation. However, both the button and label widgets have the `text` property to display text strings.

Layout

The Toolkit API service creates a widget hierarchy where the toolkit is at the top-level. The service implements group-type widgets that contain child widgets. Layout properties dictate layout actions that a parent performs on its children. For example, the `expandableX` property when set as `True`, expands the widget horizontally to encompass all of the available space. The `visible` property when set as `True` allows a widget to display in the GUI.

User Input

Some widgets allow user interaction. For example, the `textField` widget is a text box that allows user entries. Access the contents of the box with the `text` property. A Tcl script can either get or set the contents of the `textField` widget with the `text` property.

Callbacks

Some widgets perform user-specified actions, referred to as callbacks. The `textField` widget has the `onChange` property, which is called when text contents change. The `button` widget has the `onClick` property, which is called when you click a button. Callbacks update widgets or interact with services based on the contents of a text field, or the state of any other widget.

6.9.6.1 Customizing Toolkit API Widgets

Use the `toolkit_set_property` command to interact with the widgets that you instantiate. The `toolkit_set_property` command is most useful when you change part of the execution of a callback.

6.9.6.2 Toolkit API Script Examples

Example 20. Making the Toolkit Visible in System Console

Use the `toolkit_set_property` command to modify the `visible` property of the root toolkit. Use the word `self` if a property is applied to the entire toolkit. In other cases, refer to the root toolkit using `all`.

```
toolkit_set_property self visible true
```

Example 21. Adding Widgets

Use the `toolkit_add` command to add widgets.

```
toolkit_add my_button button all
```

The following commands add a label widget `my_label` to the root toolkit. In the GUI, the label appears as **Widget Label**.

```
set name "my_label"
set content "Widget Label"
toolkit_add $name label all
toolkit_set_property $name text $content
```

In the GUI, the displayed text changes to the new value. Add one more label:

```
toolkit_add my_label_2 label all
toolkit_set_property my_label_2 text "Another label"
```

The new label appears to the right of the first label.

To place the new label under the first, use the following command:

```
toolkit_set_property self itemsPerRow 1
```

Example 22. Gathering Input

To incorporate user input into your Toolkit API,

1. Create a text field using the following commands:

```
set name "my_text_field"
set widget_type "textField"
set parent "all"
toolkit_add $name $widget_type $parent
```

2. The widget size is very small. To make the widget fill the horizontal space, use the following command:

```
toolkit_set_property my_text_field expandableX true
```

3. Now, the text field is fully visible. You can type text into the field, on clicking. To retrieve the contents of the field, use the following command:

```
set content [toolkit_get_property my_text_field text]
puts $content
```

This command prints the contents into the console.

Example 23. Updating Widgets Upon User Events

When you use callbacks, the Toolkit API can also perform actions without interactive typing:

1. Start by defining a procedure that updates the first label with the text field contents:

```
proc update_my_label_with_my_text_field{
    set content [toolkit_get_property my_text_field text]
    toolkit_set_property my_label text $content
}
```

2. Run the `update_my_label_with_my_text_field` command in the Tcl Console. The first label now matches the text field contents.
3. Use the `update_my_label_with_my_text_field` command whenever the text field changes:

```
toolkit_set_property my_text_field onChange update_my_label_with_my_text_field
```

The Toolkit executes the onChange property each time the text field changes. The execution of this property changes the first field to match what you type.

Example 24. Buttons

Use buttons to trigger actions.

1. To create a button that changes the second label:

```
proc append_to_my_label_2 {suffix} {
                        set old_text [toolkit_get_property my_label_2 text]
                        set new_text "${old_text}${suffix}"
                        toolkit_set_property my_label_2 text $new_text
                        }
                        set text_to_append ", and more"
                        toolkit_add my_button button all
                        toolkit_set_property my_button onClick
[append_to_my_label_2 $text_to_append]
```

2. Click the button to append some text to the second label.

Example 25. Groups

The property `itemsPerRow` dictates the laying out of widgets in a group. For more complicated layouts where the number of widgets per row is different, use nested groups. To add a new group with more widgets per row:

```
toolkit_add my_inner_group group all
toolkit_set_property my_inner_group itemsPerRow 2
toolkit_add inner_button_1 button my_inner_group
toolkit_add inner_button_2 button my_inner_group
```

These commands create a row with a group of two buttons. To make the nested group more seamless, remove the border with the group name using the following commands:

```
toolkit_set_property my_inner_group title ""
```

You can set the `title` property to any other string to ensure the display of the border and title text.

Example 26. Tabs

Use tabs to manage widget visibility:

```
toolkit_add my_tabs tabbedGroup all
toolkit_set_property my_tabs expandableX true
toolkit_add my_tab_1 group my_tabs
toolkit_add my_tab_2 group my_tabs
toolkit_add tabbed_label_1 label my_tab_1
toolkit_add tabbed_label_2 label my_tab_2
toolkit_set_property tabbed_label_1 text "in the first tab"
toolkit_set_property tabbed_label_2 text "in the second tab"
```

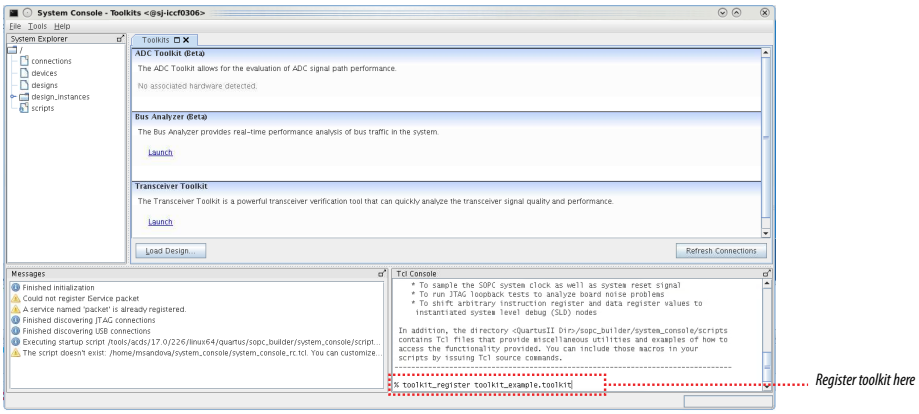
These commands add a set of two tabs, each with a group containing a label. Clicking on the tabs changes the displayed group/label.

6.9.6.3 Toolkit API GUI Example

Perform the following steps to register and launch a toolkit containing an interactive GUI window.

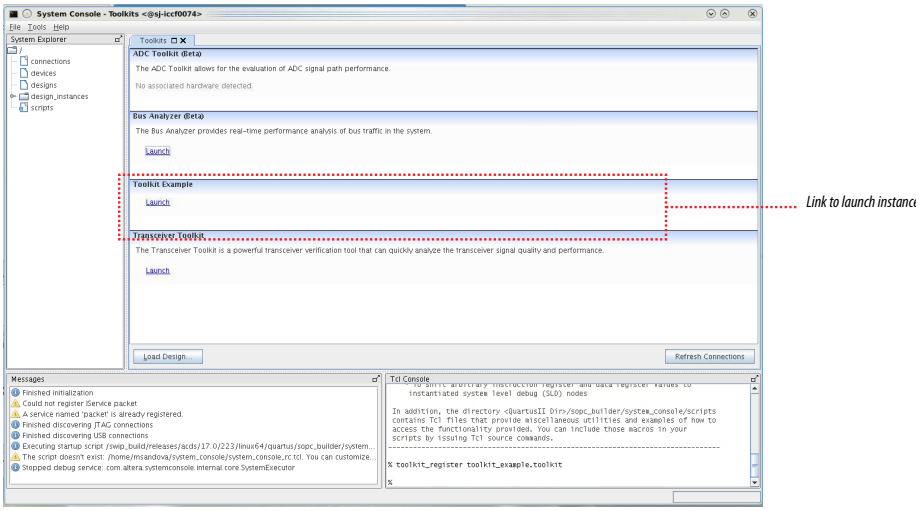
1. Write a toolkit description file. For a working example, refer to *Creating a Toolkit Description File*.
2. Generate a .tcl file using the text on *Toolkit API GUI Example .tcl File*.
3. Open the System Console.
4. Register your toolkit in the **Tcl Console** pane. Don't forget to include the relative path to your file's location.

Figure 117. Registering Your Toolkit



The Toolkit appears in the **Toolkits** tab

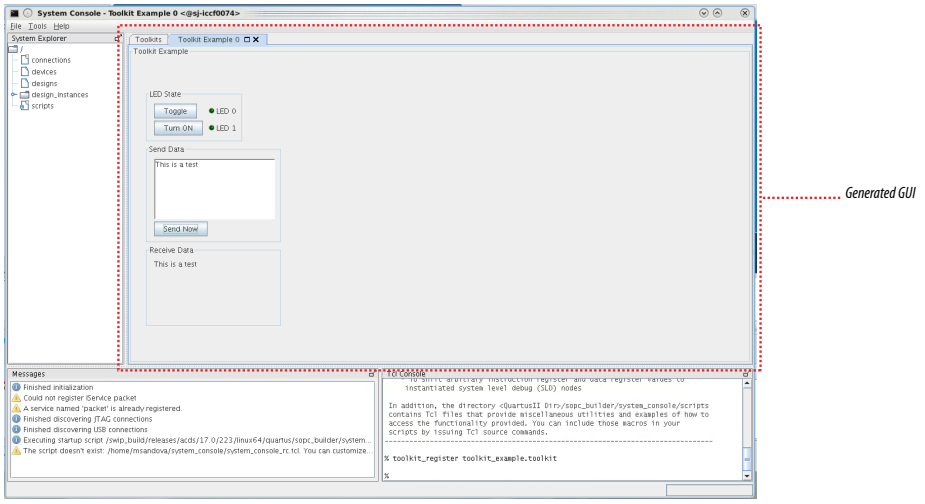
Figure 118. Toolkits Tab After Toolkit Example Registration



5. Click the Launch link.

   A new tab appears, containing the widgets you specified in the TCL file.

Figure 119. Toolkit Example GUI



When you insert text in the **Send Data** field and click **Launch**, the text appears in the **Receive Data** field.

Related Links

Creating a Toolkit Description File on page 175
A toolkit description file (`.toolkit`) is a XML file which provides the registration data for a toolkit.

6.9.6.3.1 Toolkit API GUI Example .tcl File

The following Toolkit API .tcl file creates a GUI window that provides debug interaction with your design.

```
namespace eval Test {

    variable ledValue 0
    variable dashboardActive 0
    variable Switch_off 1

    proc toggle { position } {
        set ::Test::ledValue ${position}
        ::Test::updateDashboard
        }

    proc sendText {} {
        set sendText [toolkit_get_property sendTextText text]
        toolkit_set_property receiveTextText text $sendText
        }

    proc dashBoard {} {

        if { ${::Test::dashboardActive} == 1 } {
            return -code ok "dashboard already active"
        }

        set ::Test::dashboardActive 1
        #
        # top group widget
        #
        toolkit_add  topGroup group self
        toolkit_set_property  topGroup expandableX false
        toolkit_set_property  topGroup expandableY false
```

```
toolkit_set_property  topGroup itemsPerRow 1
toolkit_set_property  topGroup title ""

#
# leds group widget
#
toolkit_add  ledsGroup group topGroup
toolkit_set_property  ledsGroup expandableX false
toolkit_set_property  ledsGroup expandableY false
toolkit_set_property  ledsGroup itemsPerRow 2
toolkit_set_property  ledsGroup title "LED State"

#
# leds widgets
#
toolkit_add  led0Button button ledsGroup
toolkit_set_property  led0Button enabled true
toolkit_set_property  led0Button expandableX false
toolkit_set_property  led0Button expandableY false
toolkit_set_property  led0Button text "Toggle"
toolkit_set_property  led0Button onClick {::Test::toggle 1}

toolkit_add  led0LED led ledsGroup
toolkit_set_property  led0LED expandableX false
toolkit_set_property  led0LED expandableY false
toolkit_set_property  led0LED text "LED 0"
toolkit_set_property  led0LED color "green_off"

toolkit_add  led1Button button ledsGroup
toolkit_set_property  led1Button enabled true
toolkit_set_property  led1Button expandableX false
toolkit_set_property  led1Button expandableY false
toolkit_set_property  led1Button text "Turn ON"
toolkit_set_property  led1Button onClick {::Test::toggle 2}

toolkit_add  led1LED led ledsGroup
toolkit_set_property  led1LED expandableX false
toolkit_set_property  led1LED expandableY false
toolkit_set_property  led1LED text "LED 1"
toolkit_set_property  led1LED color "green_off"


#
# sendText widgets
#
toolkit_add  sendTextGroup group topGroup
toolkit_set_property  sendTextGroup expandableX false
toolkit_set_property  sendTextGroup expandableY false
toolkit_set_property  sendTextGroup itemsPerRow 1
toolkit_set_property  sendTextGroup title "Send Data"

toolkit_add  sendTextText text sendTextGroup
toolkit_set_property  sendTextText expandableX false
toolkit_set_property  sendTextText expandableY false
toolkit_set_property  sendTextText preferredWidth 200
toolkit_set_property  sendTextText preferredHeight 100
toolkit_set_property  sendTextText editable true
toolkit_set_property  sendTextText htmlCapable false
toolkit_set_property  sendTextText text ""

toolkit_add  sendTextButton button sendTextGroup
toolkit_set_property  sendTextButton enabled true
toolkit_set_property  sendTextButton expandableX false
toolkit_set_property  sendTextButton expandableY false
toolkit_set_property  sendTextButton text "Send Now"
toolkit_set_property  sendTextButton onClick {::Test::sendText}

#
# receiveText widgets
#
toolkit_add  receiveTextGroup group topGroup
```

```
        toolkit_set_property  receiveTextGroup expandableX false
        toolkit_set_property  receiveTextGroup expandableY false
        toolkit_set_property  receiveTextGroup itemsPerRow 1
        toolkit_set_property  receiveTextGroup title "Receive Data"

        toolkit_add  receiveTextText text receiveTextGroup
        toolkit_set_property  receiveTextText expandableX false
        toolkit_set_property  receiveTextText expandableY false
        toolkit_set_property  receiveTextText preferredWidth 200
        toolkit_set_property  receiveTextText preferredHeight 100
        toolkit_set_property  receiveTextText editable false
        toolkit_set_property  receiveTextText htmlCapable false
        toolkit_set_property  receiveTextText text ""

        return -code ok
    }

    proc updateDashboard {} {

        if { ${::Test::dashboardActive} > 0 } {

                toolkit_set_property  ledsGroup title "LED State"
                if { [ expr ${::Test::ledValue} & 0x01 &  \
                            ${::Test::Switch_off} ] } {
                    toolkit_set_property  led0LED color "green"
                    set ::Test::Switch_off  0
                } else {
                    toolkit_set_property  led0LED color "green_off"
                    set ::Test::Switch_off  1
                }
                if { [ expr ${::Test::ledValue} & 0x02 ] } {
                    toolkit_set_property  led1LED color "green"
                } else {
                    toolkit_set_property  led1LED color "green_off"
                }
        }
    }
}
::Test::dashBoard
```

6.9.6.4 Toolkit API Commands

Toolkit API commands run in the context of a unique toolkit instance.

toolkit_register on page 185

toolkit_open on page 186

get_quartus_ini on page 187

toolkit_get_context on page 188

toolkit_get_types on page 189

toolkit_get_properties on page 190

toolkit_add on page 191

toolkit_get_property on page 192

toolkit_set_property on page 193

toolkit_remove on page 194

toolkit_get_widget_dimensions on page 195

6.9.6.4.1 toolkit_register

Description

Point to the XML file that describes the plugin (`.toolkit` file) .

Usage

`toolkit_register` <*toolkit_file*>

Returns

No return value.

Arguments

<*toolkit_file*> Path to the toolkit definition file.

Example

```
toolkit_register /path/to/toolkit_example.toolkit
```

6.9.6.4.2 toolkit_open

Description

Opens an instance of a toolkit in System Console.

Usage

`toolkit_open` <*toolkit_id*> *[<context>]*

Returns

No return value.

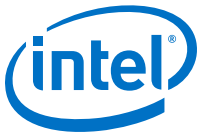
Arguments

<*toolkit_id*> Name of the toolkit type to open.

<*context*> An optional context, such as a service path for a hardware resource that is associated with the toolkit that opens.

Example

```
toolkit_open my_toolkit_id
```



6.9.6.4.3 get_quartus_ini

Description

Returns the value of an `ini` setting from the Quartus Prime software `.ini` file.

Usage

`get_quartus_ini` <*ini*> <*type*>

Returns

Value of `ini` setting.

Arguments

<*ini*> Name of the Quartus Prime software `.ini` setting.

<*type*> (Optional) Type of `.ini` setting. The known types are `string` and `enabled`. If the type is `enabled`, the value of the `.ini` setting returns `1`, or `0` if not enabled.

Example

```
set my_ini_enabled [get_quartus_ini my_ini enabled]
```

```
set my_ini_raw_value [get_quartus_ini my_ini]
```

6.9.6.4.4 toolkit_get_context

Description

Returns the context that was specified when the toolkit was opened. If no context was specified, returns an empty string.

Usage

```
toolkit_get_context
```

Returns

Context.

Arguments

No arguments.

Example

```
set context [toolkit_get_context]
```

6.9.6.4.5 toolkit_get_types

Description

Returns a list of widget types.

Usage

```
toolkit_get_types
```

Returns

List of widget types.

Arguments

No arguments.

Example

```
set widget_names [toolkit_get_types]
```

6.9.6.4.6 toolkit_get_properties

Description

Returns a list of toolkit properties for a type of widget.

Usage

`toolkit_get_properties` <*widgetType*>

Returns

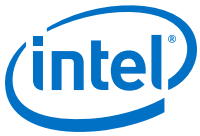
List of toolkit properties.

Arguments

<*widgetType*> Type of widget.

Example

```
set widget_properties [toolkit_get_properties xyChart]
```



6.9.6.4.7 toolkit_add

Description

Adds a widget to the current toolkit.

Usage

`toolkit_add` <*id*> <*type*><*groupid*>

Returns

No return value.

Arguments

<*id*> A unique ID for the widget being added.

<*type*> The type of widget that is being added.

<*groupid*> The ID for the parent group that will contain the new widget. Use `self` for the toolkit base group.

Example

```
toolkit_add my_button button parentGroup
```

6.9.6.4.8 toolkit_get_property

Description

Returns the property value for a specific widget.

Usage

`toolkit_get_property` <*id*> <*propertyName*>

Returns

The property value.

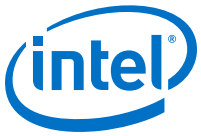
Arguments

<*id*> A unique ID for the widget being queried.

<*propertyName*> The name of the widget property.

Example

```
set enabled [toolkit_get_property my_button enabled]
```



6.9.6.4.9 toolkit_set_property

Description

Sets the property value for a specific widget.

Usage

`toolkit_set_property` <id><*propertyName*> <*value*>

Returns

No return value.

Arguments

<*id*> A unique ID for the widget being modified.

<*propertyName*> The name of the widget property being set.

<*value*> The new value for the widget property.

Example

```
toolkit_set_property my_button enabled 0
```

6.9.6.4.10 toolkit_remove

Description

Removes a widget from the specified toolkit.

Usage

`toolkit_remove <id>`

Returns

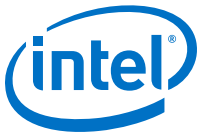
No return value.

Arguments

<id> A unique ID for the widget being removed.

Example

```
toolkit_remove my_button
```



6.9.6.4.11 toolkit_get_widget_dimensions

Description

Returns the width and height of the specified widget.

Usage

`toolkit_get_widget_dimensions <id>`

Returns

Width and height of specified widget.

Arguments

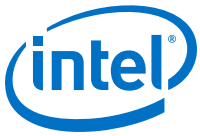
<id> A unique ID for the widget being added.

Example

```
set dimensions [toolkit_get_widget_dimensions my_button]
```

6.9.6.5 Toolkit API Properties

The following are the Toolkit API widget properties:



6.9.6.5.1 Widget Types and Properties

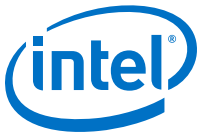
Table 44. Toolkit API Widget Types and Properties

Name	Description
`enabled`	Enables or disables the widget.
`expandable`	Controls whether the widget is expandable.
`expandableX`	Allows the widget to resize horizontally if there is space available in the cell where it resides.
`expandableY`	Allows the widget to resize vertically if there is space available in the cell where it resides.
`foregroundColor`	Sets the foreground color.
`maxHeight`	If the widget's `expandableY` is set, this is the maximum height in pixels that the widget can take.
`minHeight`	If the widget's `expandableY` is set, this is the minimum height in pixels that the widget can take.
`maxWidth`	If the widget's `expandableX` is set, this is the maximum width in pixels that the widget can take.
`minWidth`	If the widget's `expandableX` is set, this is the minimum width in pixels that the widget can take.
`preferredHeight`	The height of the widget if `expandableY` is not set.
`preferredWidth`	The width of the widget if `expandableX` is not set.
`toolTip`	Implements a mouse-over tooltip.
`visible`	Displays the widget.

6.9.6.5.2 barChart Properties

Table 45. Toolkit API barChart Properties

Name	Description
`title`	Chart title.
`labelX`	X-axis label text.
`label`	X-axis label text.
`range`	Y-axis value range. By default, it is auto range. Specifiy the range using a Tcl list, for example: `[list lower_numerical_value upper_numerical_value]`.
`itemValue`	Specify the value using a Tcl list, for example: `[list bar_category_str numerical_value]`.



6.9.6.5.3 button Properties

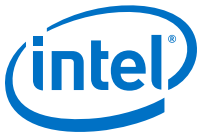
Table 46. Toolkit API button Properties

Name	Description
`onClick`	Specifies the Tcl command to run every time you click the button. Usually the command is a `proc`.
`text`	The text on the button.

6.9.6.5.4 checkBox Properties

Table 47. Toolkit API checkBox Properties

Name	Description
`checked`	Specifies the state of the checkbox.
`onClick`	Specifies the Tcl command to run every time you click the checkbox. The command is usually a `proc`.
`text`	The text on the checkbox.



6.9.6.5.5 comboBox Properties

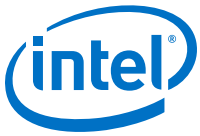
Table 48. Toolkit API comboBox Properties

Name	Description
`onChange`	A Tcl callback to run when the value of the combo box changes.
`options`	A list of items to display in the combo box.
`selectedItem`	The selected item in the combo box.

6.9.6.5.6 dial Properties

Table 49. Toolkit API dial Properties

Name	Description
`max`	The maximum value that the dial can show.
`min`	The minimum value that the dial can show.
`ticksize`	The space between the different tick marks of the dial.
`title`	The title of the dial.
`value`	The value that the dial's needle marks. It must be between min and max.



6.9.6.5.7 fileChooserButton Properties

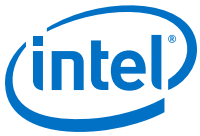
Table 50. Toolkit API fileChooserButton Properties

Name	Description
`text`	The text on the button.
`onChoose`	A Tcl command that runs every time you click the button. The command is usually a `proc`.
`title`	The title of the dialog box.
`chooserButtonText`	The text of the dialog box approval button. Default value is `Open`.
`filter`	The file filter, based on extension. The filter supports only one extension. By default, the filter allows all file names. Specify the filter using the syntax `[list filter_description file_extension]`, for example: `[list "Text Document (.txt)" "txt"]`.
`mode`	Specifies what kind of files or directories you can select. The default is `files_only`. Possible options are `files_only` and `directories_only`.
`multiSelectionEnabled`	Controls whether you can select multiple files. Default value is `false`.
`paths`	This property is read-only. Returns a list of file paths selected in the file chooser dialog box. The property is most useful when you use it within the `onClick` script, or inside a procedure that updates the result after the dialog box closes.

6.9.6.5.8 group Properties

Table 51. Toolkit API group Properties

Name	Description
`itemsPerRow`	The number of widgets the group can position in one row, from left to right, before moving to the next row.
`title`	The title of the group. Groups with a title can have a border around them, and setting an empty title removes the border.



6.9.6.5.9 label Properties

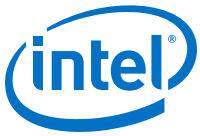
Table 52. Toolkit API label Properties

Name	Description
`text`	The text to show in the label.

6.9.6.5.10 led Properties

Table 53. Toolkit API led Properties

Name	Description
`color`	The color of the LED. The options are: `red_off`, `red`, `yellow_off`, `yellow`, `green_off`, `green`, `blue_off`, `blue`, and `black`.
`text`	The text to show next to the LED.



6.9.6.5.11 lineChart Properties

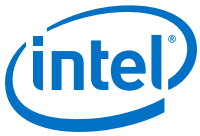
Table 54. Toolkit API lineChart Properties

Name	Description
`title`	Chart title.
`labelX`	X-axis label text.
`labelY`	Y-axis label text.
`range`	Y-axis value range. By default, it is auto range. Specify the range using a Tcl list, for example: `[list lower_numerical_value upper_numerical_value]`.
`itemValue`	Item value. Specifiy the value using a Tcl list, for example: `[list bar_category_str numerical_value]`.

6.9.6.5.12 list Properties

Table 55. Toolkit API list Properties

Name	Description
`selected`	Index of the selected item in the combo box.
`options`	List of options to display.
`onChange`	A Tcl callback to run when the selected item in the list changes.



6.9.6.5.13 pieChart Properties

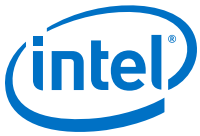
Table 56. Toolkit API pieChart Properties

Name	Description
`title`	Chart title.
`itemValue`	Item value. Specified using a Tcl list, for example: `[list bar_category_str numerical_value]`.

6.9.6.5.14 table Properties

Table 57. Toolkit API table Properties

Name	Description
`columnCount`	The number of columns (Mandatory) (`0`, by default).
`rowCount`	The number of rows (Mandatory) (`0`, by default).
`headerReorderingAllowed`	Controls whether you can drag the columns (`false`, by default).
`headerResizingAllowed`	Controls whether you can resize all column widths. (`false`, by default). *Note:* You can resize each column individually with the `columnWidthResizable` property.
`rowSorterEnabled`	Controls whether you can sort the cell values in a column (`false`, by default).
`showGrid`	Controls whether to draw both horizontal and vertical lines (`true`, by default).
`showHorizontalLines`	Controls whether to draw horizontal line (`true`, by default).
`rowIndex`	Current row index. Zero-based. This value affects some properties below (`0`, by default).
`columnIndex`	Current column index. Zero-based. This value affects all column specific properties below (`0`, by default).
`cellText`	Specifies the text inside the cell given by the current `rowIndex` and `columnIndex` (Empty, by default).
`selectedRows`	Control or retrieve row selection.
`columnHeader`	The text in the column header.
`columnHeaders`	A list of names to define the columns for the table.
`columnHorizontalAlignment`	The cell text alignment in the specified column. Supported types are `leading` (default), `left`, `center`, `right`, `trailing`.
`columnRowSorterType`	The type of sorting method. This is applicable only if `rowSorterEnabled` is `true`. Each column has its own sorting type. Possible types are `string` (default), `int`, and `float`.
`columnWidth`	The number of pixels in the column width.
`columnWidthResizable`	Controls whether the column width is resizable by you (`false`, by default).
`contents`	The contents of the table as a list. For a table with columns A, B, and C, the format of the list is `{A1 B1 C1 A2 B2 C2 ...}`.



6.9.6.5.15 text Properties

Table 58. Toolkit API text Properties

Name	Description
`editable`	Controls whether the text box is editable.
`htmlCapable`	Controls whether the text box can format HTML.
`text`	The text to show in the text box.

6.9.6.5.16 textField Properties

Table 59. Toolkit API textField Properties

Name	Description
`editable`	Controls whether the text box is editable.
`onChange`	A Tcl callback to run when you change the content of the text box.
`text`	The text in the text box.

6.9.6.5.17 timeChart Properties

Table 60. Toolkit API timeChart Properties

Name	Description
`labelX`	The label for the X-axis.
`labelY`	The label for the Y-axis.
`latest`	The latest value in the series.
`maximumItemCount`	The number of sample points to display in the historic record.
`title`	The title of the chart.
`range`	Sets the range for the chart. The range has the form `{low, high}`. The low/high values are doubles.
`showLegend`	Specifies whether a legend for the series is shown in the graph.

6.9.6.5.18 xyChart Properties

Table 61. Toolkit API xyChart Properties

Name	Properties
`title`	Chart title.
`labelX`	X-Axis label text.
`labelY`	Y-Axis label text.
`range`	Sets the range for the chart. The range is of the form `{low, high}`. The low/high values are doubles.
`maximumItemCount`	Specifies the maximum number of data values to keep in a data series. This setting only affects new data in the chart. If you add more data values than the `maximumItemCount`, only the last `maximumItemCount` number of entries are kept.
`series`	Adds a series of data to the chart. The first value in the spec is the identifier for the series. If the same identifier is set twice, the Toolkit API selects the most recent series. If the identifier does not contain series data, that series is removed from the chart. Specify the series in a Tcl list: `{identifier, x-1 y-1, x-2 y-2}`.
`showLegend`	Sets whether a legend for the series appears in the graph.

6.10 System Console Examples and Tutorials

Intel provides examples for performing board bring-up, creating a simple dashboard, and programming a Nios II processor. The `System_Console.zip` file contains design files for the board bring-up example. The Nios II Ethernet Standard `.zip` files contain the design files for the Nios II processor example.

Note: The instructions for these examples assume that you are familiar with the Quartus Prime software, Tcl commands, and Qsys Pro.

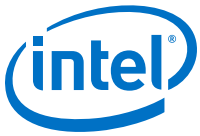
Related Links

On-Chip Debugging Design Examples Website
Contains the design files for the example designs that you can download.

6.10.1 Board Bring-Up with System Console Tutorial

You can perform low-level hardware debugging of Qsys Pro systems with System Console. You can debug systems that include IP cores instantiated in your Qsys Pro system or perform initial bring-up of your PCB. This board bring-up tutorial uses a Nios II Embedded Evaluation Kit (NEEK) board and USB cable. If you have a different development kit, you need to change the device and pin assignments to match your board and then recompile the design.

1. Setting Up the Board Bring-Up Design Example on page 215
   To load the design example into the Quartus Prime software and program your device, perform the following steps:
2. Verifying Clock and Reset Signals on page 216
   You can use the System Explorer pane to verify clock and reset signals.
3. Verifying Memory and Other Peripheral Interfaces on page 216



The Avalon-MM service accesses memory-mapped slaves via a suitable Avalon-MM master, which can be controlled by the host.

4. Qsys Pro Modules for Board Bring-up Example on page 221

Related Links

Introduction to System Console on page 156
System Console provides visibility into your design and allows you to perform system-level debug on a FPGA at run-time. System Console performs tests on debug-enabled Qsys Pro instantiated IP cores. A variety of debug services provide read and write access to elements in your design. You can perform the following tasks with System Console and the tools built on top of System Console:

6.10.1.1 Setting Up the Board Bring-Up Design Example

To load the design example into the Quartus Prime software and program your device, perform the following steps:

1. Unzip the `System_Console.zip` file to your local hard drive.
2. In the Quartus Prime software, click **File ➤ Open Project** and select `Systemconsole_design_example.qpf`.
3. Change the device and pin assignments (LED, clock, and reset pins) in the `Systemconsole_design_example.qsf` file to match your board.
4. Click **Processing ➤ Start Compilation**
5. To program your device, follow these steps:
   a. Click **Tools ➤ Programmer**.
   b. Click **Hardware Setup**.
   c. Click the **Hardware Settings** tab.
   d. Under **Currently selected hardware**, click **USB-Blaster**, and click **Close**.

      Note: If you do not see the **USB-Blaster** option, then your device was not detected. Verify that the USB-Blaster driver is installed, your board is powered on, and the USB cable is intact.

      This design example uses a USB-Blaster cable. If you do not have a USB-Blaster cable and you are using a different cable type, then select your cable from the **Currently selected hardware** options.

   e. Click **Auto Detect**, and then select your device.
   f. Double-click your device under **File**.
   g. Browse to your project folder and click `Systemconsole_design_example.sof` in the subdirectory **output_files**.
   h. Turn on the **Program/Configure** option.
   i. Click **Start**.
   j. Close the Programmer.
6. Click **Tools** > **System Debugging Tools** > **System Console**.

Related Links

System_Console.zip file

Contains the design files for this tutorial.

6.10.1.2 Verifying Clock and Reset Signals

You can use the System Explorer pane to verify clock and reset signals.

Open the appropriate node and check for either a green clock icon or a red clock icon. You can use JTAG Debug command to verify clock and reset signals.

Related Links

- System Explorer Pane on page 161
  The **System Explorer** pane displays the virtual file system for all connected debugging IP cores, and contains the following information:
- JTAG Debug Commands on page 174

6.10.1.3 Verifying Memory and Other Peripheral Interfaces

The Avalon-MM service accesses memory-mapped slaves via a suitable Avalon-MM master, which can be controlled by the host. You can use Tcl commands to read and write to memory with a master service.

6.10.1.3.1 Locating and Opening the Master Service

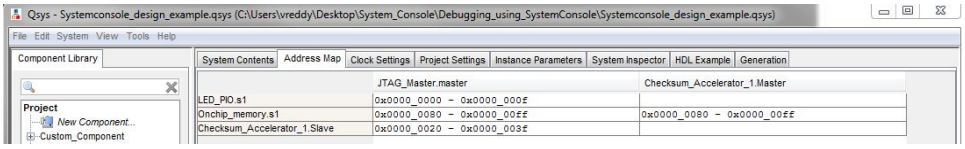
```
#Select the master service type and check for available service paths.
set service_paths [get_service_paths master]

#Set the master service path.
set master_service_path [lindex $service_paths 0]

#Open the master service.
set claim_path [claim_service master $master_service_path mylib]
```

6.10.1.3.2 Avalon-MM Slaves

The **Address Map** tab shows the address range for every Qsys Pro component. The Avalon-MM master communicates with slaves using these addresses.

The register maps for all Intel FPGA components are in their respective Data Sheets.

Figure 120. Address Map



Related Links

Data Sheets Website

Avalon-MM Commands

You can read or write the Avalon-MM interfaces using the master read and write commands. You can also use the master commands on slave services. If you are working on a slave service, the address field can be a register. [6]

Table 62. Avalon-MM Commands

Command	Arguments	Function
master_write_memory	<service-path> <address> <list_of_byte_values>	Writes the specified list of byte values to the specified service path and address.
master_write_8	<service-path> <address> <list_of_byte_values>	Writes the specified list of byte values to the specified service path and address, using 8-bit accesses.
master_write_16	<service-path> <address> <list_of_16_bit_words>	Writes the specified list of 16-bit values to the specified service path and address, using 16-bit accesses.
master_write_from_file	<service-path> <file-name> <address>	Writes the entire contents of the file to the specified service path and address. The file is treated as a binary file containing a stream of bytes.
master_write_32	<service-path> <address> <list_of_32_bit_words>	Writes the specified list of 32-bit values to the specified service path and address, using 32-bit accesses.
master_read_memory	[-format <format>] <service-path> <address> <size_in_bytes>	Returns a list of read values in bytes. Memory read starts at the specified base address. *Note:* The [-format <format>] is an optional argument. Specifying this argument makes this command accept data as 16 or 32-bit, instead as bytes. For example: `master_read_memory -format 16 <service_path> <addr> <count>`
master_read_8	<service-path> <address> <size_in_bytes>	Returns a list of <size> bytes. Read from memory starts at the specified base address, using 8-bit accesses.
master_read_16	<service-path> <address> <size_in_multiples_of_16_bits>	Returns a list of <size> 16-bit values. Read from memory starts at the specified base address, using 16-bit accesses.
master_read_32	<service-path> <address> <size_in_multiples_of_32_bits>	Returns a list of <size> 32-bit values. Read from memory starts at the specified base address, using 32-bit accesses.
master_read_to_file	<service-path> <file-name> <address> <count>	Reads the number of bytes specified by <count> from the memory address specified and creates (or overwrites) a file containing the values read. The file is written as a binary file.
master_get_register_names	<service-path>	When a register map is defined, returns a list of register names in the slave.

Note: Using the 8, 16, or 32 versions of the `master_read` or `master_write` commands is less efficient than using the `master_write_memory` or `master_read_memory` commands.

6 Transfers performed in 16- and 32-bit sizes are packed in little-endian format.

6.10.1.3.3 Testing the PIO component

In this example design, the PIO connects to the LEDs of the board. Test if this component is operating properly and the LEDs are connected, by driving the outputs with the Avalon-MM master.

Table 63. Register Map for the PIO Core

Offset	Register Name		R/W	Fields				
				(n-1)	...	2	1	0
0	data	read access	R	Data value currently on PIO inputs.				
		write access	W	New value to drive on PIO outputs.				
1	direction		R/W	Individual direction control for each I/O port. A value of 0 sets the direction to input; 1 sets the direction to output.				
2	interruptmask		R/W	IRQ enable/disable for each input port. Setting a bit to 1 enables interrupts for the corresponding port.				
3	edgecapture		R/W	Edge detection for each input port.				

```
#Write the driver output values for the Parallel I/O component.
set offset 0x0; #Register address offset.
set value 0x7; #Only set bits 0, 1, and 2.
master_write_8 $claim_path $offset $value

#Read back the register value.
set offset 0x0
set count 0x1
master_read_8 $claim_path $offset $count

master_write_8 $claim_path 0x0 0x2; #Only set bit 1.

master_write_8 $claim_path 0x0 0xe; #Only set bits 1, 2, 3.

master_write_8 $claim_path 0x0 0x7; #Only set bits 0, 1, 2.

#Observe the LEDs turn on and off as you execute these Tcl commands.
#The LED is on if the register value is zero and off if the register value is one.
#LED 0, LED 1, and LED 2 connect to the PIO.
#LED 3 connects to the interrupt signal of the CheckSum Accelerator.
```

6.10.1.3.4 Testing On-chip Memory

Test the memory with a recursive function that writes to incrementing memory addresses.

```
#Load the design example utility procedures for writing to memory.
source set_memory_values.tcl

#Write to the on-chip memory.
set base_address 0x80
set write_length 0x80
set value 0x5a5a5a5a
fill_memory $claim_path $base_address $write_length $value

#Verify the memory was written correctly.
#This utility proc returns 0 if the memory range is not uniform with this value.
verify_memory $claim_path $base_address $write_length $value

#Check that the memory is re-initialized when reset.
#Trigger reset then observe verify_memory returns 0.
set jtag_debug_path [lindex [get_service_paths jtag_debug] 0]
```

```
set claim_jtag_debug_path [claim_service jtag_debug $jtag_debug_path mylib]
jtag_debug_reset_system $claim_jtag_debug_path; #Reset the connected on-chip
memory
#peripheral.
close_service jtag_debug $claim_jtag_debug_path
verify_memory $claim_path $base_address $write_length $value

#The on-chip memory component was parameterized to re-initialized to 0 on reset.
#Check the actual value.
master_read_8 $claim_path 0x0 0x1
```

6.10.1.3.5 Testing the Checksum Accelerator

The Checksum Accelerator calculates the checksum of a data buffer in memory. It calculates the value for a specified memory buffer, sets the DONE bit in the status register, and asserts the interrupt signal. You should only read the result from the controller when both the DONE bit and the interrupt signal are asserted. The host should assert the interrupt enable control bit in order to check the interrupt signal.

Table 64. Register Map for Checksum Component

Offset (Bytes)	Hexadecimal value (after adding offset)	Register	Access	Bits (32 bits)							
				31-9	8	7-5	4	3	2	1	0
0	0x20	Status	Read/Write to clear							BUSY	DONE
4	0x24	Address	Read/Write	Read Address							
12	0x2C	Length	Read/Write	Length in bytes							
24	0x38	Control	Read/Write		Fixed Read Address Bit		Interrupt Enable	GO		INV	Clear
28	0x3C	Result	Read	Checksum result (upper 16 bits are zero)							

1.

```
#Pass the base address of the memory buffer Checksum Accelerator.
set base_address 0x20
set offset 4
set address_reg [expr {$base_address + $offset}]
set memory_address 0x80
master_write_32 $claim_path $address_reg $memory_address

#Pass the memory buffer to the Checksum Accelerator.
set length_reg [expr {$base_address + 12}]
set length 0x20
master_write_32 $claim_path $length_reg $length

#Write clear to status and control registers.
#Status register:
set status_reg $base_address
master_write_32 $claim_path $status_reg 0x0
#Control register:
set clear 0x1
set control_reg [expr {$base_address + 24}]
master_write_32 $claim_path $control_reg $clear

#Write GO to the control register.
set go 0x8
master_write_32 $claim_path $control_reg $go

#Cross check if the checksum DONE bit is set.
master_read_32 $claim_path $status_reg 0x1
```

```
#Is the DONE bit set?
#If yes, check the result and you are finished with the board bring-up design
example.
set result_reg [expr {$base_address + 28}]
master_read_16 $claim_path $result_reg 0x1
```

2. If the result is zero and the JTAG chain works properly, the clock and reset signals work properly, and the memory works properly, then the problem is the Checksum Accelerator component.

```
#Confirm if the DONE bit in the status register (bit 0)
#and interrupt signal are asserted.
#Status register:
master_read_32 $claim_path $status_reg 0x1
#Check DONE bit should return a one.

#Enable interrupt and go:
set interrupt_and_go 0x18
master_write_32 $claim_path $control_reg $interrupt_and_go
```

3. Check the control enable to see the interrupt signal. LED 3 (MSB) should be off. This indicates the interrupt signal is asserted.
4. You have narrowed down the problem to the data path. View the RTL to check the data path.
5. Open the `Checksum_transform.v` file from your project folder.
   - *<unzip dir>*`/System_Console/ip/checksum_accelerator/checksum_accelerator.v`
6. Notice that the `data_out` signal is grounded in Figure 121 on page 220 (uncommented line 87 and comment line 88). Fix the problem.
7. Save the file and regenerate the Qsys Pro system.
8. Re-compile the design and reprogram your device.
9. Redo the above steps, starting with Verifying Memory and Other Peripheral Interfaces on page 216 or run the Tcl script included with this design example.

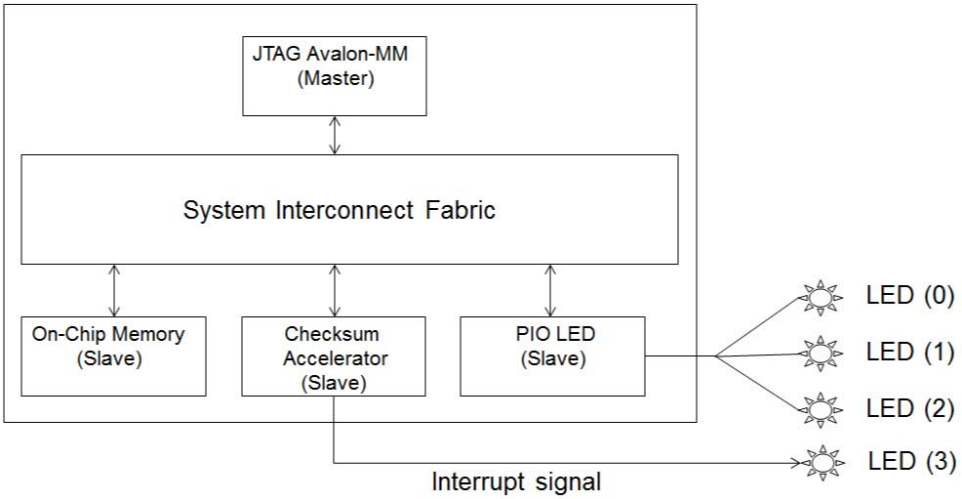
```
source set_memory_and_run_checksum.tcl
```

Figure 121. Checksum.v File

```
83      // first folding
84      assign first_folded_sum = (initial_sum [32] + initial_sum[31:16] + initial_sum[15:0]);  // this result is at most 17 bits wide (16 bits with rollover)
85
86      // second folding and optional inversion, this result is at most 16 bits wide
87      assign data_out = (invert == 1)? ~(first_folded_sum[16] + first_folded_sum[15:0]) : (first_folded_sum[16] + first_folded_sum[15:0]);
88    // assign data_out = 16'h0000;
89
90    endmodule
```

6.10.1.4 Qsys Pro Modules for Board Bring-up Example

Figure 122. Qsys Pro Modules for Board Bring-up Example



The Qsys Pro design for this example includes the following modules:

- JTAG to Avalon Master Bridge—Provides System Console host access to the memory-mapped IP in the design via the JTAG interface.
- On-chip memory—Simplest type of memory for use in an FPGA-based embedded system. The memory is implemented on the FPGA; consequently, external connections on the circuit board are not necessary.
- Parallel I/O (PIO) module—Provides a memory-mapped interface for sampling and driving general I/O ports.
- Checksum Accelerator—Calculates the checksum of a data buffer in memory. The Checksum Accelerator consists of the following:
  - Checksum Calculator (`checksum_transform.v`)
  - Read Master (`slave.v`)
  - Checksum Controller (`latency_aware_read_master.v`)

6.10.1.4.1 Checksum Accelerator Functionality

The base address of the memory buffer and data length passes to the Checksum Controller from a memory-mapped master. The Read Master continuously reads data from memory and passes the data to the Checksum Calculator. When the checksum calculations finish, the Checksum Calculator issues a valid signal along with the checksum result to the Checksum Controller. The Checksum Controller sets the DONE bit in the status register and also asserts the interrupt signal. You should only read the result from the Checksum Controller when the DONE bit and interrupt signal are asserted.

6.10.2 Nios II Processor Example

This example programs the Nios II processor on your board to run the count binary software example included in the Nios II installation. This is a simple program that uses an 8-bit variable to repeatedly count from 0x00 to 0xFF. The output of this

variable is displayed on the LEDs on your board. After programming the Nios II processor, you use System Console processor commands to start and stop the processor.

To run this example, perform the following steps:

1. Download the Nios II Ethernet Standard Design Example for your board from the Altera website.
2. Create a folder to extract the design. For this example, use `C:\Count_binary`.
3. Unzip the Nios II Ethernet Standard Design Example into `C:\Count_binary`.
4. In a Nios II command shell, change to the directory of your new project.
5. Program your board. In a Nios II command shell, type the following:

```
nios2-configure-sof niosii_ethernet_standard_<board_version>.sof
```

6. Using Nios II Software Build Tools for Eclipse, create a new Nios II Application and BSP from Template using the **Count Binary** template and targeting the Nios II Ethernet Standard Design Example.
7. To build the executable and linkable format (ELF) file (`.elf`) for this application, right-click the **Count Binary** project and select **Build Project**.
8. Download the `.elf` file to your board by right-clicking **Count Binary** project and selecting **Run As, Nios II Hardware**.
   - The LEDs on your board provide a new light show.
9. Type the following:

```
system-console; #Start System Console.

#Set the processor service path to the Nios II processor.
set niosii_proc [lindex [get_service_paths processor] 0]

set claimed_proc [claim_service processor $niosii_proc mylib]; #Open the
service.

processor_stop $claimed_proc; #Stop the processor.
#The LEDs on your board freeze.

processor_run $claimed_proc; #Start the processor.
#The LEDs on your board resume their previous activity.

processor_stop $claimed_proc; #Stop the processor.

close_service processor $claimed_proc; #Close the service.
```

   - The `processor_step`, `processor_set_register`, and `processor_get_register` commands provide additional control over the Nios II processor.

Related Links

- Nios II Ethernet Standard Design Example
- Nios II Gen2 Software Developer's Handbook

6.10.2.1 Processor Commands

Table 65. Processor Commands

Command [7]	Arguments	Function
`processor_download_elf`	*<service-path>* *<elf-file-path>*	Downloads the given Executable and Linking Format File (`.elf`) to memory using the master service associated with the processor. Sets the processor's program counter to the `.elf` entry point.
`processor_in_debug_mode`	*<service-path>*	Returns a non-zero value if the processor is in debug mode.
`processor_reset`	*<service-path>*	Resets the processor and places it in debug mode.
`processor_run`	*<service-path>*	Puts the processor into run mode.
`processor_stop`	*<service-path>*	Puts the processor into debug mode.
`processor_step`	*<service-path>*	Executes one assembly instruction.
`processor_get_register_names`	*<service-path>*	Returns a list with the names of all of the processor's accessible registers.
`processor_get_register`	*<service-path>* *<register_name>*	Returns the value of the specified register.
`processor_set_register`	*<service-path>* *<register_name>* *<value>*	Sets the value of the specified register.

Related Links

Nios II Processor Example on page 221
This example programs the Nios II processor on your board to run the count binary software example included in the Nios II installation.

6.11 On-Board USB Blaster II Support

System Console supports an On-Board USB-Blaster™ II circuit via the USB Debug Master IP component. This IP core supports the master service.

Not all Stratix V boards support the On-Board USB-Blaster II. For example, the transceiver signal integrity board does not support the On-Board USB-Blaster II.

6.12 About Using MATLAB and Simulink in a System Verification Flow

System Console can be used with MATLAB and Simulink to perform system development testing. You can use the Intel FPGA Hardware in the Loop (HIL) tools to set up a system verification flow. In this approach, the design is deployed to hardware and runs in real time. The surrounding components in your system are simulated in a software environment. The HIL approach allows you to use the flexibility of software tools with the real-world accuracy and speed of hardware. You can gradually introduce

7 If your system includes a Nios II/f core with a data cache, it may complicate the debugging process. If you suspect the Nios II/f core writes to memory from the data cache at nondeterministic intervals; thereby, overwriting data written by the System Console, you can disable the cache of the Nios II/f core while debugging.

more hardware components to your system verification testbench. This gives you more control over the integration process as you tune and validate your system. When your full system is integrated, the HIL approach allows you to provide stimuli via software to test your system under a variety of scenarios.

Advantages of HIL Approach

- Avoid long computational delays for algorithms with high processing rates
- API helps to control, debug, visualize, and verify FPGA designs all within the MATLAB environment
- FPGA results are read back by the MATLAB software for further analysis and display

Required Tools and Components

- MATLAB software
- DSP Builder for Intel® FPGAs software
- Quartus Prime software
- Intel FPGA

Note: The System Console MATLAB API is included in the DSP Builder for Intel FPGAs installation bundle.

Figure 123. Hardware in the Loop Host-Target Setup

Supported MATLAB API Commands

You can perform your work from the MATLAB environment and leverage the capability of System Console to read and write to masters and slaves. By using the supported MATLAB API commands, you do not have to launch the System Console software. The supported commands are the following:

- `SystemConsole.refreshMasters;`
- `M = SystemConsole.openMaster(1);`
- `M.write (type, byte address, data);`
- `M.read (type, byte address, number of words);`
- `M.close`

Example 27. MATLAB API Script Example

```
SystemConsole.refreshMasters; %Investigate available targets
M = SystemConsole.openMaster(1); %Creates connection with FPGA target
%%%%%%%% User Application %%%%%%%%%%%%
....
M.write('uint32',write_address,data); %Send data to FPGA target
....
data = M.read('uint32',read_address,size); %Read data from FPGA target
....
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
M.close; %Terminates connection to FPGA target
```

High-Level Flow

1. Install the DSP Builder for Intel FPGAs software so you have the necessary libraries to enable this flow
2. Build your design using Simulink and the DSP Builder for Intel FPGAs libraries (DSP Builder for Intel FPGAs helps to convert the Simulink design to HDL)
3. Include Avalon-MM components in your design (DSP Builder for Intel FPGAs can port non-Avalon-MM components)
4. Include Signals and Control blocks in your design
5. Use boundary blocks to separate synthesizable and non-synthesizable logic
6. Integrate your DSP system in Qsys Pro
7. Program your Intel FPGA
8. Use the supported MATLAB API commands to interact with your Intel FPGA

Related Links

Hardware in the Loop from the MATLAB/Simulink Environment white paper

6.13 Deprecated Commands

The table lists commands that have been deprecated. These commands are currently supported, but are targeted for removal from System Console.

Note: All `dashboard_<name>` commands are deprecated and replaced with `toolkit_<name>` commands for Quartus Prime software15.1, and later.

Table 66. Deprecated Commands

Command	Arguments	Function
`open_service`	*<service_type>* *<service_path>*	Opens the specified service type at the specified path. Calls to `open_service` may be replaced with calls to `claim_service` providing that the return value from `claim_service` is stored and used to access and close the service.

6.14 Document Revision History

Table 67. Document Revision History

Date	Version	Changes
2017.05.08	17.0.0	• Created topic *Convert your Dashboard Scripts to Toolkit API*. • Removed *Registering the Service* Example from *Toolkit API Script Examples*, and added corresponding code snippet to *Registering a Toolkit*. • Moved *.toolkit Description File Example* under *Creating a Toolkit Description File*. • Renamed *Toolkit API GUI Example .toolkit File* to *.toolkit Description File Example*. • Updated examples on Toolkit API to reflect current supported syntax.
2016.10.31	16.1.0	• Implemented Intel rebranding.
2015.11.02	15.1.0	• Edits to Toolkit API content and command format. • Added Toolkit API design example. • Added graphic to *Introduction to System Console*. • Deprecated Dashboard. • Changed instances of *Quartus II* to *Quartus Prime*.
October 2015	15.1.0	• Added content for Toolkit API — Required .toolkit and Tcl files — Registering and launching the toolkit — Toolkit discovery and matching toolkits to IP — Toolkit API commands table
May 2015	15.0.0	Added information about how to download and start System Console stand-alone.
December 2014	14.1.0	• Added overview and procedures for using ADC Toolkit on MAX 10 devices. • Added overview for using MATLAB/Simulink Environment with System Console for system verification.
June 2014	14.0.0	Updated design examples for the following: board bring-up, dashboard service, Nios II processor, design service, device service, monitor service, bytestream service, SLD service, and ISSP service.
November 2013	13.1.0	Re-organization of sections. Added high-level information with block diagram, workflow, SLD overview, use cases, and example Tcl scripts.
June 2013	13.0.0	Updated Tcl command tables. Added board bring-up design example. Removed SOPC Builder content.
November 2012	12.1.0	Re-organization of content.
August 2012	12.0.1	Moved Transceiver Toolkit commands to Transceiver Toolkit chapter.
June 2012	12.0.0	Maintenance release. This chapter adds new System Console features.
November 2011	11.1.0	Maintenance release. This chapter adds new System Console features.
May 2011	11.0.0	Maintenance release. This chapter adds new System Console features.
December 2010	10.1.0	Maintenance release. This chapter adds new commands and references for Qsys.
July 2010	10.0.0	Initial release. Previously released as the System Console User Guide, which is being obsoleted. This new chapter adds new commands.

Related Links

Altera Documentation Archive

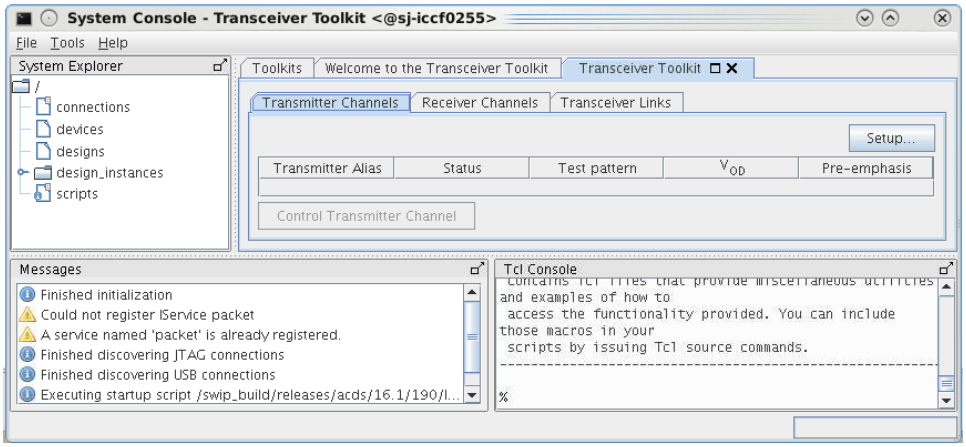
For previous versions of the *Quartus Prime Handbook*, search the Altera documentation archives.

7 Debugging Transceiver Links

The Transceiver Toolkit helps you to optimize high-speed serial links in your board design. The Transceiver Toolkit provides real-time control, monitoring, and debugging of the transceiver links running on your board.

Once you correctly configure a debugging system, you can control the transmitter or receiver channels to optimize transceiver settings and hardware features. The toolkit tests bit-error rate (BER) while running multiple links at target data rate. Run auto sweep tests to identify the best physical media attachment (PMA) settings for each link. The toolkit supports testing of multiple devices across multiple boards simultaneously.

Figure 124. Transceiver Toolkit



Transceiver Toolkit User Interface

- **System Explorer**—displays design components and hardware connections.
- **Channel Manager**—control multiple channels simultaneously.
- **Tcl Console**—script control of the Transceiver Toolkit.
- **Messages**—displays information, warning, and error messages.

Quick Start

The Transceiver Toolkit user interface helps you to visualize and debug transceiver links in your design. To launch the toolkit, click **Tools ➤ System Debugging Tools ➤ Transceiver Toolkit**. Alternatively, you can run Tcl scripts from the command-line:

```
system-console --script=<name of script>
```

Get started quickly by downloading Transceiver Toolkit design examples from the On-Chip Debugging Design Examples website. For an online demonstration of how to use the Transceiver Toolkit to run a high-speed link test with one of the design examples, refer to the Transceiver Toolkit Online Demo on the Altera website.

Related Links

- On-Chip Debugging Design Examples
- Transceiver Toolkit Online Demo

7.1 Transceiver Debugging Flow

Testing transceiver links involves configuring your system for debug, and then running various link tests.

Table 68. Transceiver Link Debugging Flow

	Flow Description
System Configuration Steps	1. Use one of the following methods to define a system that includes necessary transceiver debugging components: • Download Transceiver Toolkit design examples from the On-Chip Debugging Design Examples website. Modify Intel FPGA design examples to fit your design. • Integrate debugging components into your own design. Click **Tools ➤ IP Catalog** to select IP cores and define them in the parameter editor. 2. Click **Assignments ➤ Pin Planner** to assign device I/O pins to match your device and board. 3. Click **Tools ➤ BluePrint Platform Designer ➤** to plan a legal device periphery floorplan. 4. Click **Processing ➤ Start Compilation** to compile your design. 5. Connect your target device to Intel programming hardware. 6. Click **Tools ➤ Programmer** and then program the target device with the debugging system.
Link Debugging Steps	1. Click **File ➤ Load Design**, and select the SRAM Object File (`.sof`) generated for your transceiver design. If you start the toolkit while a project is open, the project loads in the toolkit automatically. 2. (Optional) Create additional links between transmitter and receiver channels. 3. Run any of the following tests: • Run BER with various combinations of PMA settings. • Run custom traffic tests. • Run link optimization tests. • Directly control PMA analog settings to experiment with settings while the link is running.

Related Links

BluePrint Platform Planning

7.2 Configuring Systems for Transceiver Debug

To debug transceivers, you must first configure a system that includes the appropriate Intel FPGA IP core(s) that support each debugging operation. You can either modify an Intel FPGA design example, or configure your own system with required debugging IP components.The debugging system configuration varies by device family. Refer to the appropriate setup information to correctly configure or adapt a system for your target device.

7.2.1 Configuring an Intel FPGA Design Example

Intel provides design examples to help you quickly test your own design. You can experiment with Intel FPGA design examples and modify them for your own application. Refer to the `readme.txt` of each design example for more information. Download the Transceiver Toolkit design examples from the On-Chip Debugging Design Examples page of the Intel FPGA website.

Use the design examples as a starting point to work with a particular signal integrity development board. The design examples provide the components to quickly test the functionality of the receiver and transmitter channels in your design. Change the transceiver settings in the design examples and observe the effects on transceiver link performance. Isolate and verify the high-speed serial links without debugging other logic in your design. You can modify and customize the design examples to match your intended transceiver design.

Once you download the design examples, open the Quartus Prime and click **Project ➤ Restore Archived Project** to restore the design example project archive. If you have access to the same development board with the same device as mentioned in the `readme.txt` file of the example, you can directly program the device with the provided programming file in that example. If you want to recompile the design, you must make your modifications to the system configuration in Qsys Pro, regenerate in Qsys Pro, and recompile the design in the Quartus Prime software to generate a new programming file.

If you have the same board as mentioned in the `readme.txt` file, but a different device on your board, you must choose the appropriate device and recompile the design. For example, some early development boards are shipped with engineering sample devices.

You can make changes to the design examples so that you can use a different development board or a different device. If you have a different board, you must edit the necessary pin assignments and recompile the design examples.

Related Links

On-Chip Debugging Design Examples

7.2.2 Configuring Your Own Debugging System

Rather than modifying the Intel FPGA Debugging Design Examples, you can integrate debugging IP components into your own design. Refer to the appropriate system configuration steps for your target device.

Related Links

Arria 10 Debug System Configuration on page 230
For Arria 10 designs, the Transceiver Toolkit configuration requires instantiation of the Arria 10 Transceiver Native PHY IP core, and use of the built-in Intel FPGA Debug Master Endpoint (ADME).

7.2.2.1 Arria 10 Debug System Configuration

For Arria 10 designs, the Transceiver Toolkit configuration requires instantiation of the Arria 10 Transceiver Native PHY IP core, and use of the built-in Intel FPGA Debug Master Endpoint (ADME). Qsys Pro system generation automatically instantiates the JTAG debug link. You enable the ADME features by enabling specific parameters in the Transceiver Native PHY IP core.

Click **Tools ➤ IP Catalog** to parameterize, generate, and instantiate the following debugging components.

Table 69. Arria 10 / 20nm Transceiver Toolkit IP Core Configuration

Component	Debugging Functions	Parameterization Notes
Transceiver Native PHY	Supports all debugging functions	On the **Dynamic Reconfiguration** tab: • Turn on **Enable dynamic reconfiguration**. • Turn on **Enable odi acceleration logic**. • Turn on **Enable Intel FPGA Debug Master Endpoint**. • Turn on **Enable capability registers**. • Turn on **Enable control and status registers**. • Turn on **Enable PRBS soft accumulators** (enables hard PRBS Generator and Checker).
Transceiver ATX PLL	Required for Arria 10	On the **Dynamic Reconfiguration** tab: • Turn on **Enable dynamic reconfiguration** (required for System Console read/write access). • Turn on **Enable Intel FPGA Debug Master Endpoint** (required for System Console read/write access).
Transceiver PHY Reset Controller	N/A	N/A
Intel FPGA Debug Master Endpoint (ADME)	• Supports control of PMA analog settings, ADCE settings, and DFE settings. • discovers PHY and use toolkit on designs not using Qsys Pro. • Optionally, turn off to save resource count. (only an option if you instantiate the JTAG to Avalon Master Bridge. Otherwise, Transceiver Toolkit cannot function).	• Enabled from the Arria 10 Transceiver Native PHY Parameter Editor, **Dynamic Reconfiguration** tab.
JTAG Debug Link	Required for Arria 10.	For Qsys Pro projects, the JTAG Debug Link is auto-instantiated.

7.2.2.1.1 Enabling Intel FPGA Debug Master Endpoint (Arria 10)

The Intel FPGA Debug Master Endpoint (ADME) connects to the host link via the system level debug fabric for debugging Arria 10 designs. The ADME provides Avalon-MM Master capability, and is discoverable by System Console. To enable ADME for Arria 10 designs, you must set specific parameters when defining the Arria 10 Transceiver Native PHY and Arria 10 Transceiver ATX PLL IP cores. Click **Tools ➤ IP Catalog** to select and define the following IP core parameters. Once properly configured, project synthesis inserts the ADME, debug fabric, and embedded logic.

Figure 125. Intel FPGA Debug Master Endpoint Block Diagram

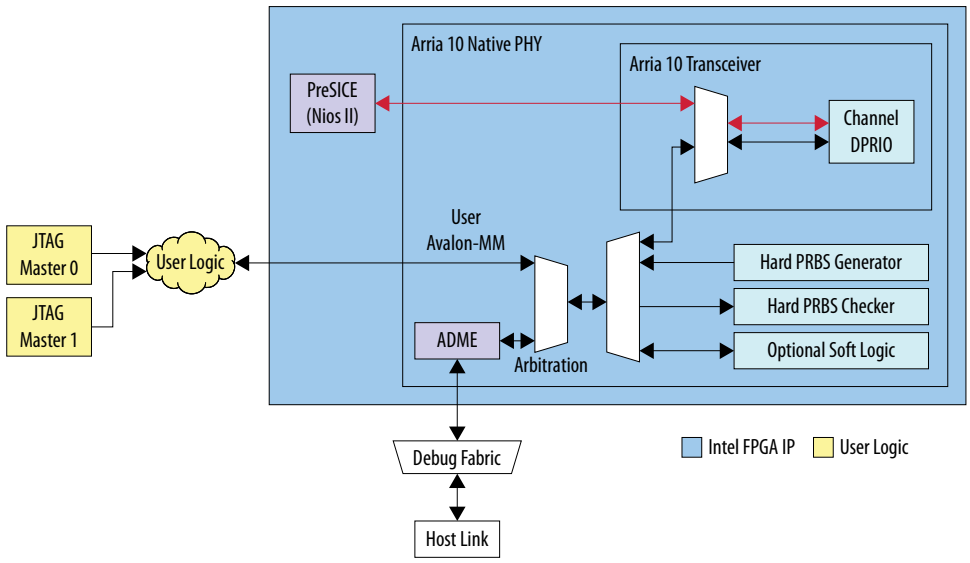


Table 70. Enabling ADME IP Core Parameters for Arria 10 Designs

IP Core	Parameter Setting	Location
Arria 10 Transceiver Native PHY	• Turn on **Enable dynamic reconfiguration** • Turn on **Enable Intel FPGA Debug Master Endpoint** • Turn on **Enable prbs soft accumulators**	**Dynamic Reconfiguration** tab
Arria 10 Transceiver ATX PLL	• Turn on **Enable capability registers** • Turn on **Enable control and status registers**	**Dynamic Reconfiguration** tab

Figure 126. Enabling ADME in Arria 10 Transceiver Native PHY IP Core

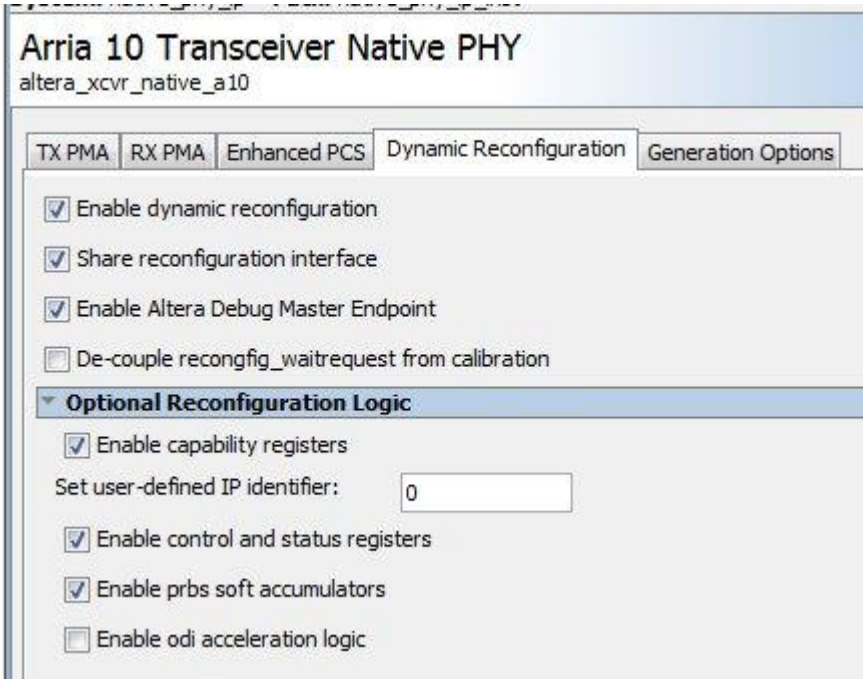
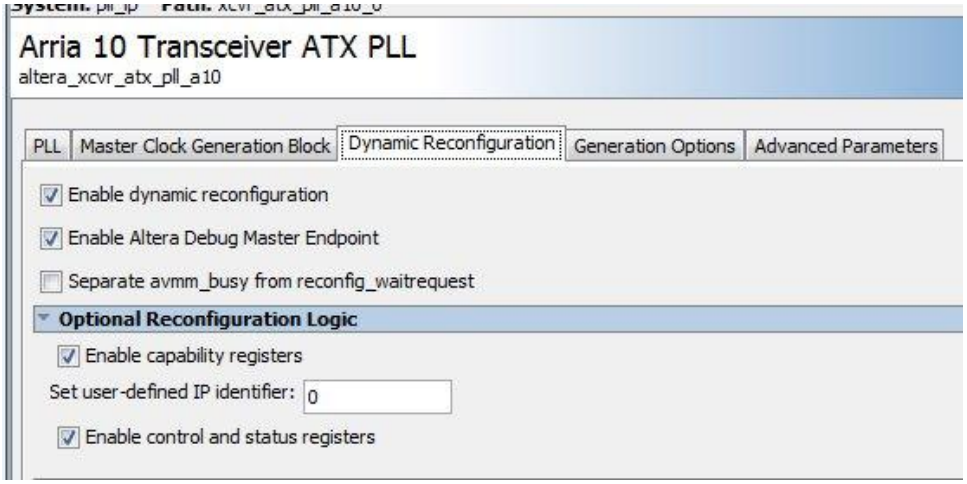


Figure 127. Enabling ADME in ATX PLL IP Core



7.2.2.1.2 Link Testing Configuration (Arria 10)

Use the following system configuration for BER, PRBS, and link optimization testing in Arria 10 devices. To perform BER or PRBS testing of Arria 10 designs, use the PRBS Generator and PRBS Checker functions of the Transceiver Native PHY IP core. You enable the built-in hard PRBS Generator and Checker by turning on **Enable prbs soft accumulators** when configuring the IP core. The Transceiver Toolkit performs required read-write-modify operations on the tested channel(s).

The built-in hard PRBS Data Pattern Generator and Checker requires no separate IP instantiation. The hard PRBS Data Pattern Generator and Checker does not support error injection. To use error injection for Arria 10 designs, use the Data Pattern Generator and Checker IP cores in your Arria 10 system design configuration.

Figure 128. BER, PRBS, and Link Optimization Test Configuration (Arria 10)

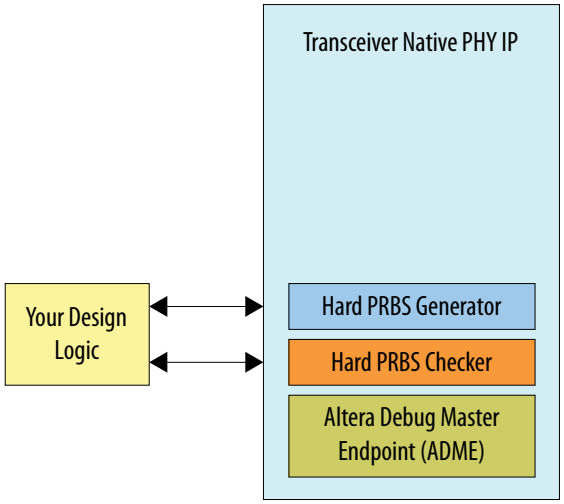


Table 71. System Connections: BER and PRBS Testing (Arria 10)

From	To
Your design logic	• Transceiver Native PHY Hard PRBS Generator • Transceiver Native PHY Hard PRBS Checker

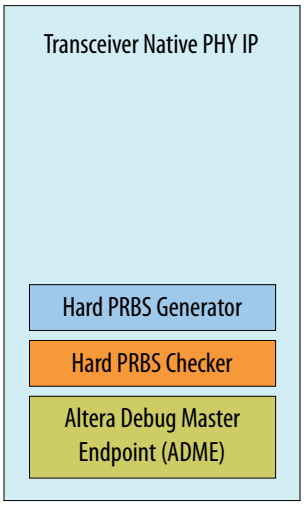
Related Links

Using PRBS and Square Wave Data Pattern Generator and Checker
In the *Arria 10 Transceiver Native PHY User Guide.*

7.2.2.1.3 PMA Analog Setting Control Configuration (Arria 10)

Use the following configuration to control PMA analog settings in Arria 10 devices.

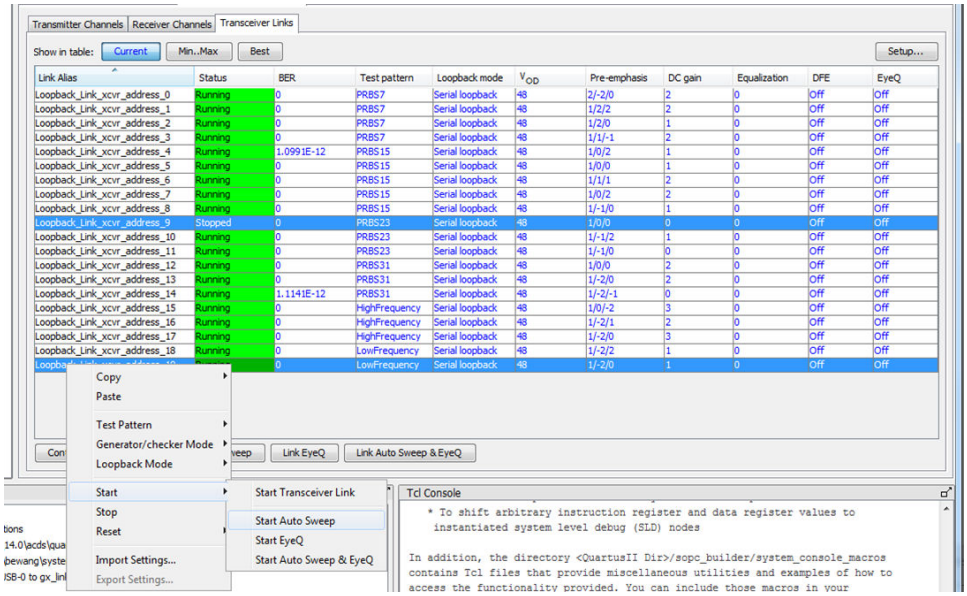
Figure 129. System Configuration: PMA Analog Setting Control (Arria 10)

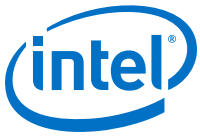


7.3 Managing Transceiver Channels

Configure and control large numbers of channels in a spreadsheet view using the **Channel Manager**. View all the PMA and sweep settings for all channels. Copy, paste, import, and export settings to and from channels. You can also start and stop sweeps for any or all channels. Right-click the **Channel Manager** to view additional channel commands. The columns in the **Channel Manager** are movable, resizable, and sortable.

Figure 130. Channel Manager GUI





Copying and Pasting Settings

You can copy PMA and/or sweep settings from a selected row. You can paste PMA and/or sweep settings to one or more rows.

Importing and Exporting Settings

Select a row in the **Channel Manager** to export your PMA settings to a text file. Select one or more rows in the **Channel Manager** to apply the PMA settings from a text file. The PMA settings in the text file apply to a single channel. When you import the PMA settings from a text file, you are duplicating one set of PMA settings for all selected channels.

Starting and Stopping Tests

The **Channel Manager** allows you to start and stop tests by right-clicking the channels. You can select several rows in the **Channel Manager** to start or stop test for multiple channels.

7.3.1 Channel Display Modes

The three display modes are **Current**, **Min/Max**, and **Best**. The default display mode is **Current**.

- **Current**—shows the current values from the device. The blue color text indicates that the settings are live.
- **Min/Max**—shows the minimum and maximum values to be used in the auto sweep.
- **Best**—shows the best tested values from the last completed auto sweep run.

Note: The **Transmitter Channels** tab only shows the **Current** display mode. Auto sweep cannot be performed on only a transmitter channel; a receiver channel is required to perform an auto sweep test.

7.3.2 Creating Links

The toolkit automatically creates links when a receiver and transmitter share a transceiver channel. You can also manually create and delete links between transmitter and receiver channels. You create links in the **Setup** dialog.

Setup Dialog

Click **Setup** from the **Channel Manager** to open the **Setup** dialog box.

Table 72. Setup Dialog Popup Menu

Command Name	Action When Clicked	Enabled If
Edit Transmitter Alias	Starts the inline edit of the alias of the selected row.	Only enabled if one row is selected.
Edit Receiver Alias	Starts the inline edit of the alias of the selected row.	Only enabled if one row is selected.
Edit Transceiver Link Alias	Starts the inline edit of the alias of the selected row.	Only enabled if one row is selected.
Copy	Copies the text of the selected row(s) to the clipboard. The text copied depends on the column clicked on. The text copied to the clipboard is newline delimited.	Enabled if one or more rows are selected.

7.3.3 Controlling Transceiver Channels

You can directly control and monitor transmitters, receivers, and links running on the board in real time.You can transmit a data pattern across the transceiver link, and then report the signal quality of the received data.

Click **Control Transmitter Channel** (**Transmitter Channels** tab), **Control Receiver Channel** (**Receiver Channels** tab), or **Control Transceiver Link** (**Transceiver Links** tab) to adjust transmitter or receiver settings while the channels are running.

7.4 Debugging Transceiver Links

The Transceiver Toolkit allows you to control and monitor the performance of high-speed serial links running on your board in real-time.

You can identify the transceiver links in your design, transmit a data pattern across the transceiver link, and report the signal quality of the received data in terms of bit error rate.

The toolkit automatically identifies the transceiver links in your design, or you can manually create transceiver links. Run auto sweep to quickly identify the best PMA settings for each link. You can directly control the transmitter/receiver channels to experiment with various settings suggested by auto sweep.

The Transceiver Toolkit supports various transceiver link testing configurations. Identify and test the transceiver link between two Intel FPGA devices, or transmit a test pattern with a third-party device and monitor the data on an Intel FPGA device receiver channel. If a third-party chip includes self-test capability, send the test pattern from the Intel FPGA device and monitor the signal integrity at the third-party device receiver channel. If the third-party device supports reverse serial loopback, run the test entirely within the Transceiver Toolkit.

Before you can monitor transceiver channels, you must configure a system with debugging components, and program the design into an FPGA. Once those steps are complete, use the following flow to test the channels:

1. Load the design in Transceiver Toolkit
2. Link hardware resources
3. Verify hardware connections
4. Identify transceiver channels
5. Run link tests or control PMA analog settings
6. View results

7.4.1 Step 1: Load Your Design

To load any design into the toolkit, click **File ➤ Load Design** and select the `.sof` programming file generated for your transceiver design. The Transceiver Toolkit automatically loads the last compiled design upon opening. Loading the `.sof` automatically links the design to the target hardware in the toolkit. The toolkit automatically discovers links between transmitter and receiver of the same channel. The **System Explorer** displays information about the loaded design.

7.4.2 Step 2: Link Hardware Resources

The toolkit automatically discovers connected hardware and designs. You can also manually link a design to connected hardware resources in the **System Explorer**.

If you are using more than one Intel FPGA board, you can set up a test with multiple devices linked to the same design. This setup is useful when you want to perform a link test between a transmitter and receiver on two separate devices. You can also load multiple Quartus Prime projects and make links between different systems. You can perform tests on completely separate and unrelated systems in a single tool instance.

Note: Prior to the Transceiver Toolkit version 11.1, you must manually load and link your design to hardware. In version 11.1 and later, the Transceiver Toolkit automatically links any device programmed with a project.

Figure 131. One Channel Loopback Mode for Arria 10 / 20nm

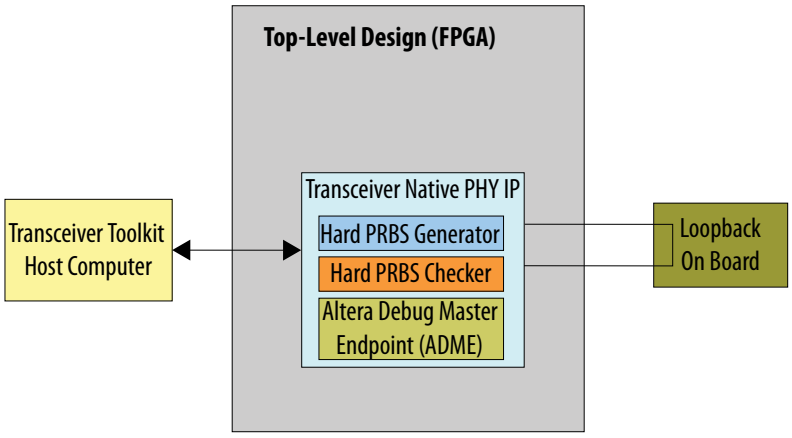
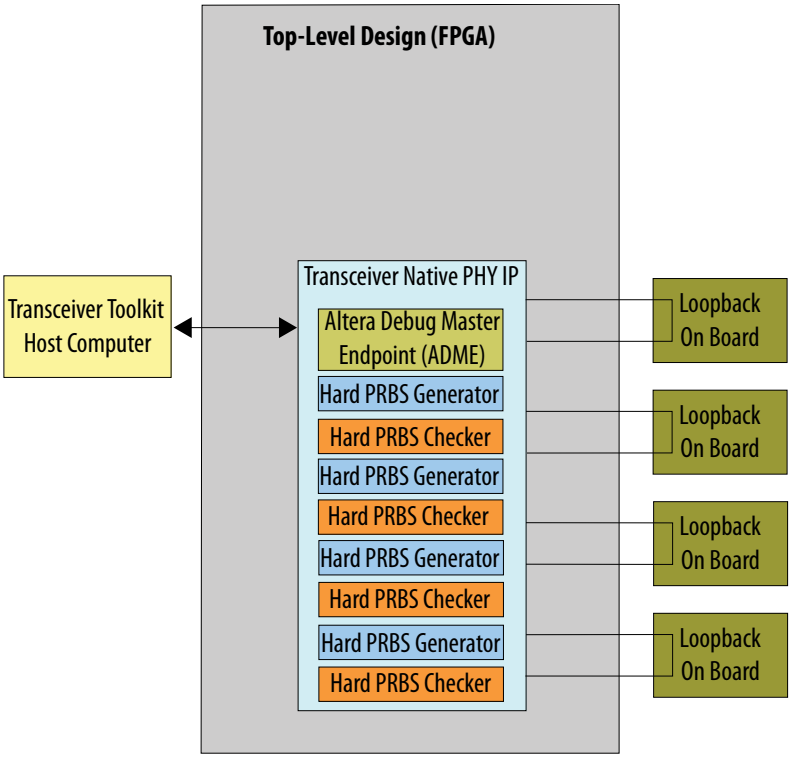


Figure 132. Four Channel Loopback Mode for Arria 10 / Generation 10 / 20nm



7.4.2.1 Linking One Design to One Device

To link one design to one device by one USB-Blaster download cable, follow these steps:

1. Load the design for your Quartus Prime project.
2. Link each device to an appropriate design if the design has not auto-linked.
3. Create the link between channels on the device to test.

7.4.2.2 Linking Two Designs to Two Devices

To link two designs to two separate devices on the same board, connected by one USB-Blaster download cable, follow these steps:

1. Load the design for all the Quartus Prime project files you might need.
2. Link each device to an appropriate design if the design has not auto-linked.
3. Open the project for the second device.
4. Link the second device on the JTAG chain to the second design (unless the design auto-links).
5. Create a link between the channels on the devices you want to test.

7.4.2.3 Linking Designs and Devices on Separate Boards

To link two designs to two separate devices on separate boards, connected to separate USB-Blaster download cables, follow these steps:

1. Load the design for all the Quartus Prime project files you might need.
2. Link each device to an appropriate design if the design has not auto-linked.
3. Create the link between channels on the device to test.
4. Link the device you connected to the second USB-Blaster download cable to the second design.
5. Create a link between the channels on the devices you want to test.

7.4.2.4 Linking One Design on Two Devices

To link the same design on two separate devices, follow these steps:

1. In the Transceiver Toolkit, open the `.sof` you are using on both devices.
2. Link the first device to this design instance.
3. Link the second device to the design.
4. Create a link between the channels on the devices you want to test.

7.4.3 Step 3: Verify Hardware Connections

After you load your design and link your hardware, verify that the channels are connected correctly and looped back properly on the hardware. Use the toolkit to send data patterns and receive them correctly. Verifying hardware connections before you perform link tests can save time in the work flow.

After you have verified that the transmitter and receiver are communicating with each other, you can create a link between the two transceivers so that you can perform Auto Sweep tests with this pair.

7.4.4 Step 4: Identify Transceiver Channels

The Transceiver Toolkit automatically displays recognized transmitter and receiver channels. The toolkit identifies a channel automatically whenever a receiver and transmitter share a transceiver channel. You can also manually identify the transmitter and receiver in a transceiver channel and create a link between the two for testing.

When you run link tests, channel color highlights indicate the test status:

Table 73. Channel Color Highlights

Color	Transmitter Channel	Receiver Channel
Red	Channel is closed or generator clock is not running	Channel is closed or checker clock is not running
Green	Generator is sending a pattern	Checker is checking and data pattern is locked
Neutral	Channel is open, generator clock is running, and generator is not sending a pattern	Channel is open, checker clock is running, and checker is not checking
Yellow	N/A	Checker is checking and data pattern is not locked

7.4.5 Step 5: Run Link Tests

Once you identify the transceiver channels for debugging, you can run various link tests in the toolkit.

Use the **Transceiver Links** tab to control link tests. For example, use the Auto Sweep feature to sweep transceiver settings to determine the parameters that support the best BER value. Click **Link Auto Sweep** to adjust the PMA settings and run tests.

7.4.5.1 Running BER Tests

Run BER tests across your transceiver link after programming the FPGA with your debugging design, loading the design in the toolkit, and linking hardware.Follow these steps to run BER tests:

1. Click **Setup**.
   a. Select the generator and checker you want to control.
   b. Select the transmitter and receiver pair you want to control.
   c. Click **Create Transceiver Link** and click **Close**
2. Click **Control Transceiver Link**, and specify a PRBS **Test pattern** and **Data pattern checker** for **Checker mode**. The checker mode option is only available after you turn on **Enable Bit Error Rate Block** in the Reconfiguration Controller component.

   If you select **Bypass** for the **Test pattern**, the toolkit bypasses the PRBS generator and runs your design through the link. The bypass option is only available after you turn on **Enable Bypass interface** in the Reconfiguration Controller component.
3. Experiment with **Reconfiguration**, **Generator**, or **Checker** settings.
4. Click **Start** to run the pattern with your settings. You can then click **Inject Error** to inject error bits, **Reset** the counter, or **Stop** the test.

   Note: Arria 10 devices do not support **Inject Error** if you use the hard PRBS Pattern Generator and Checker in the system configuration.

Related Links

Link Testing Configuration (Arria 10) on page 232
Use the following system configuration for BER, PRBS, and link optimization testing in Arria 10 devices.

7.4.5.2 Auto Sweep Testing

Use the auto sweep feature to automatically sweep ranges for the best transceiver PMA settings. Store a history of the test runs and keep a record of the best PMA settings. Use the best settings the toolkit determines in your final design for improved signal integrity.

7.4.5.2.1 Running Link Optimization Tests

After programming the FPGA with your debugging design, loading the design in the toolkit, and linking hardware, follow these steps to run link optimization tests:

1. Click the **Transceiver Links** tab, and select the channel you want to control.
2. Click **Link Auto Sweep**. The **Advanced** tab appears with **Auto sweep** as **Test mode**.
3. Specify the PRBS **Test pattern**.
4. Specify **Run length**, experiment with the **Transmitter settings**, and **Receiver settings** to control the test coverage and PMA settings, respectively.
5. Click **Start** to run all combinations of tests meeting the PMA parameter limits.
6. When the run completes the chart is displayed and the characteristics of each run are listed in the run list. You can click **Stop** to halt the test, change the PMA settings, and re-start the test. Click **Create Report** to export data to a table format for further viewing.
7. To use decision feedback equalization (DFE) to determine the best tap settings, follow these steps:
   a. Use Auto Sweep to find optimal PMA settings while leaving the **DFE mode** set to **Off**.
   b. If BER = 0, use the best PMA settings achieved.
   c. If BER > 0, use this PMA setting, and set the minimum and maximum values obtained from Auto Sweep to match this setting. Set the maximum DFE range to limits for each of the three DFE settings.
   d. Run **Create Report** to view the results and determine which DFE setting has the best BER. Use these settings in conjunction with the PMA settings for the best results.

Related Links

Link Testing Configuration (Arria 10) on page 232
Use the following system configuration for BER, PRBS, and link optimization testing in Arria 10 devices.

7.4.6 Controlling PMA Analog Settings

You can directly control PMA analog settings to experiment with settings while the link is running. To control PMA analog settings, follow these steps:

1. Click **Setup**.
   a. Click the **Transmitter Channels** tab, define a transmitter without a generator, and click **Create Transmitter Channel**.
   b. Click the **Receiver Channels** tab, define a receiver without a generator, and click **Create Receiver Channel**.
   c. Click the **Transceiver Links** tab, select the transmitter and receivers you want to control, and click **Create Transceiver Link**.
   d. Click **Close**.
2. Click **Control Receiver Channel**, **Control Transmitter Channel**, or **Control Transceiver Link** to directly control the PMA settings while running.

Figure 133. Controlling Transmitter Channel

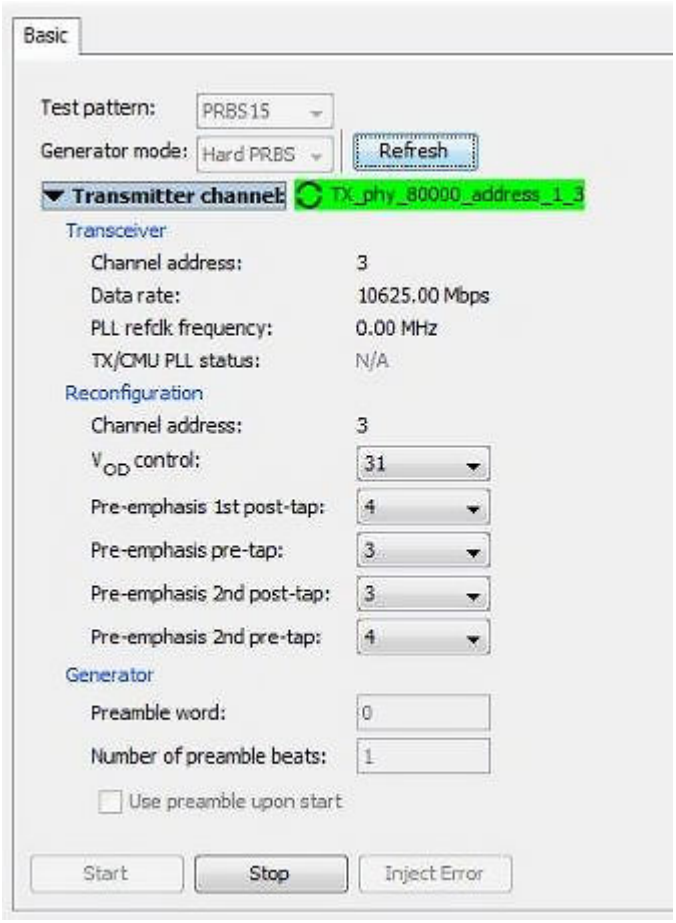


Figure 134. Controlling Receiver Channel

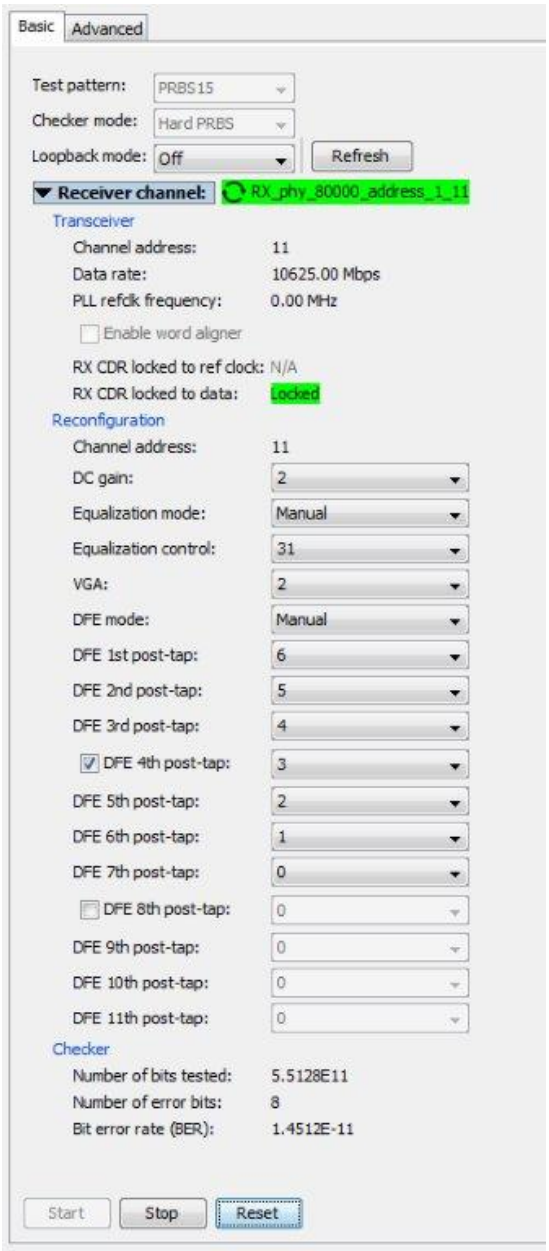
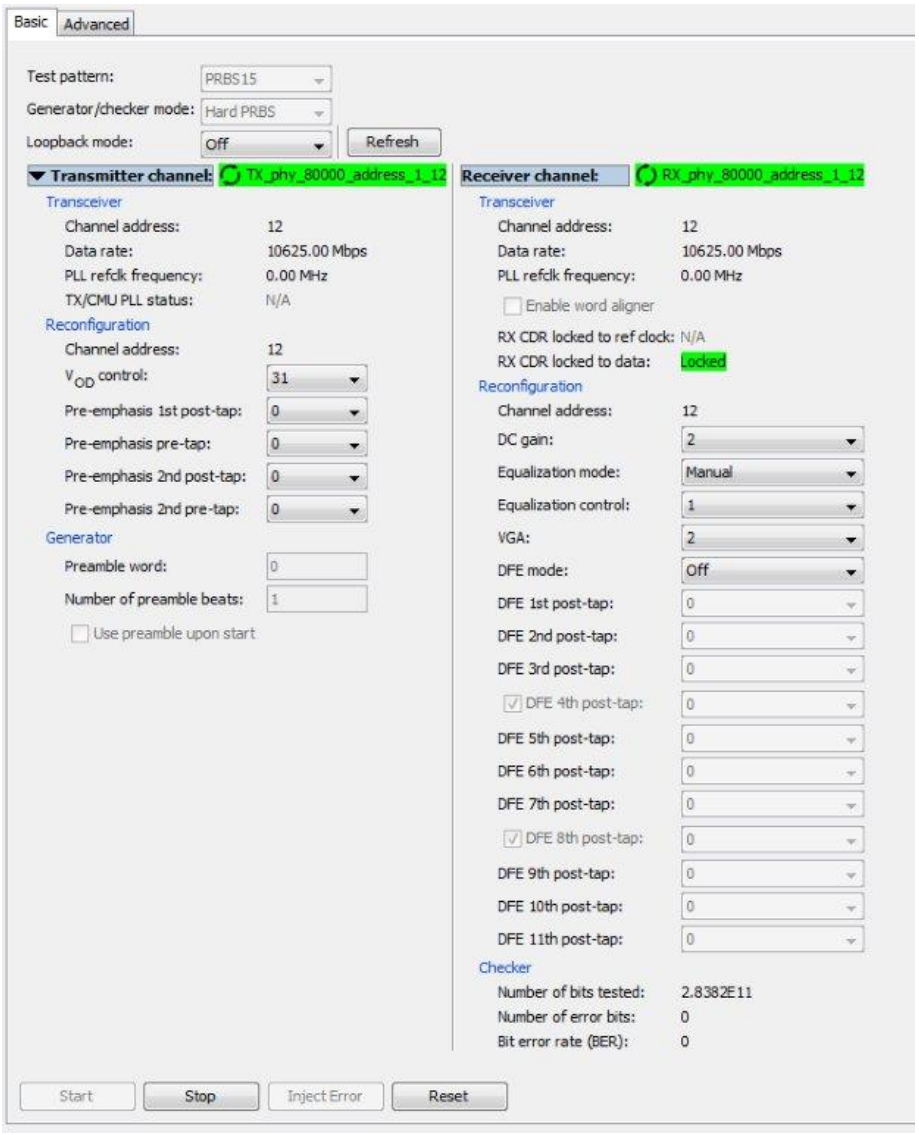


Figure 135. Controlling Transceiver Link



Related Links

PMA Analog Setting Control Configuration (Arria 10) on page 234
Use the following configuration to control PMA analog settings in Arria 10 devices.

7.5 Troubleshooting Common Errors

- Missing high-speed link pin connections

  The pin connections to identify high-speed links (tx_p/n and rx_p/n) could be missing. When porting an older design to the latest version of the Quartus Prime software, please make sure your connections were preserved.

- Reset Issues

  Ensure that the reset input to the Transceiver Native PHY, Transceiver Reset Controller, and ATX PLL IP cores is not held active (1'b1). The Transceiver Toolkit highlights in red, all the Transceiver Native PHY channels which you are trying to set up.

- Unconnected `reconfig_clk`

  The `reconfig_clk` input to the Transceiver Native PHY and ATX PLL IP cores need to be connected and driven. Otherwise, the transceiver link channel will not be displayed.

7.6 User Interface Settings Reference

The following settings are available for interaction with the transmitter channels or receiver channels or transceiver links in the Transceiver Toolkit user interface.

Table 74. Transceiver Toolkit Control Panel Settings

Setting	Description	Control Panel
Alias	Name you choose for the channel.	Transmitter Receiver Transceiver Link
Auto Sweep status	Reports the current and best tested bits, errors, bit error rate, and case count for the current Auto Sweep test.	Receiver Transceiver Link
Bit error rate (BER)	Specifies errors divided by bits tested since the last reset of the checker.	Receiver Transceiver Link
Channel address	Logical address number of the transceiver channel.	Transmitter Receiver Transceiver Link
Checker mode	Specify **Data pattern checker** or **Serial bit comparator** for BER tests. If you enable **Serial bit comparator** the Data Pattern Generator sends the PRBS pattern, but the pattern is checked by the serial bit comparator. In **Bypass mode**, clicking **Start** begins counting on the Serial bit comparator. For BER testing: • Arria 10 devices support the Data Pattern Checker and the Hard PRBS.	Receiver Transceiver Link
Data rate	Data rate of the channel as read from the project file or data rate as measured by the frequency detector. To use the frequency detector, turn on **Enable Frequency Counter** in the Data Pattern Checker IP core and/or Data Pattern Generator IP core, regenerate the IP cores, and recompile the design. The measured data rate depends on the Avalon management clock frequency as read from the project file. Click the refresh button next to the measured **Data rate** if you make changes to your settings and want to sample the data rate again.	Transmitter Receiver Transceiver Link
DC gain	Circuitry that provides an equal boost to the incoming signal across the frequency spectrum.	Receiver Transceiver Link
DFE mode	Decision feedback equalization (DFE) for improving signal quality.	Receiver

continued...

Setting	Description	Control Panel
• Values 1-11 (Arria 10 devices)	In Arria 10 devices DFE modes are **Off**, **Manual** and **Adaptation Enabled**. DFE in **Adaptation Enabled** mode automatically tries to find the best tap values.	Transceiver Link
Equalization control	Boosts the high-frequency gain of the incoming signal, thereby compensating for the low-pass filter effects of the physical medium. When used with DFE, use DFE in **Manual** or **Adaptation Enabled** mode.	Receiver Transceiver Link
Equalization mode	You can set **Equalization Mode** to **Manual** or **Triggered** for Arria 10 devices.	Receiver Transceiver Link
Error rate limit	Turns on or off error rate limits. **Start checking after** waits until the set number of bits are satisfied until it starts looking at the bit error rate (BER) for the next two checks. **Bit error rate achieves below** sets upper bit error rate limits. If the error rate is better than the set error rate, the test ends. **Bit error rate exceeds** Sets lower bit error rate limits. If the error rate is worse than the set error rate, the test ends.	Receiver Transceiver Link
Increase test range	Right-click the **Advanced** panel to use the span capabilities of Auto Sweep to automatically increase the span of tests by one unit down for the minimum and one unit up for the maximum, for the selected set of controls. You can span either PMA Analog controls (non-DFE controls), or the DFE controls. You can quickly set up a test to check if any PMA setting combinations near your current best could yield better results.	Receiver Transceiver Link
Maximum tested bits	Sets the maximum number of tested bits for each test iteration.	Receiver Transceiver Link
Number of bits tested	Specifies the number of bits tested since the last reset of the checker.	Receiver Transceiver Link
Number of error bits	Specifies the number of error bits encountered since the last reset of the checker.	Receiver Transceiver Link
PLL refclk freq	Channel reference clock frequency as read from the project file or measured reference clock frequency as calculated from the measured data rate.	Transmitter Receiver Transceiver Link
Populate with	Right-click the **Advanced** panel to load current values on the device as a starting point, or initially load the best settings determined through Auto Sweep. The Quartus Prime software automatically applies the values you specify in the drop-down lists for the Transmitter settings and Receiver settings.	Receiver Transceiver Link
Preamble word	Word to send out if you use the preamble mode (only if you use soft PRBS Data Pattern Generator and Checker).	Transmitter Transceiver Link
Pre-emphasis	The programmable pre-emphasis module in each transmit buffer boosts high frequencies in the transmit data signal, which may be attenuated in the transmission media.	Transmitter Transceiver Link
Receiver channel	Specifies the name of the selected receiver channel.	Receiver Transceiver Link
Refresh Button	After loading the `.pof`, loads fresh settings from the registers after running dynamic reconfiguration.	Transmitter Receiver Transceiver Link
Reset	Resets the current test.	Receiver Transceiver Link

continued...

Setting	Description	Control Panel
Rules Based Configuration (RBC) validity checking	Displays invalid combination of settings in red in each list under **Transmitter settings** and **Receiver settings**, based on previous settings. If selected, the settings remain in red to indicate the currently selected combination is invalid. This avoids manually testing invalid settings that you cannot compile for your design. This prevents setting the device into an invalid mode for extended periods of time and potentially damaging the circuits.	Receiver Transceiver Link
Run length	Sets coverage parameters for test runs.	Transmitter Receiver Transceiver Link
RX CDR PLL status	Shows the receiver in lock-to-reference (LTR) mode. When in auto-mode, if data cannot be locked, this signal alternates in LTD mode if the CDR is locked to data.	Receiver Transceiver Link
RX CDR data status	Shows the receiver in lock-to-data (LTD) mode. When in auto-mode, if data cannot be locked, the signal stays high when locked to data and never toggles.	Receiver Transceiver Link
Serial loopback enabled	Inserts a serial loopback before the buffers, allowing you to form a link on a transmitter and receiver pair on the same physical channel of the device.	Transmitter Receiver Transceiver Link
Start	Starts the pattern generator or checker on the channel to verify incoming data.	Transmitter Receiver Transceiver Link
Stop	Stops generating patterns and testing the channel.	Transmitter Receiver Transceiver Link
Test mode	Allows you to specify the test mode. On Quartus Prime, Arria 10 supports **Auto Sweep** test mode.	Receiver Transceiver Link
Test pattern	Test pattern sent by the transmitter channel. Arria 10 devices support **PRBS9**, **PRBS15**, **PRBS23**, and **PRBS31**).	Transmitter Receiver Transceiver Link
Time limit	Specifies the time limit unit and value to have a maximum bounds time limit for each test iteration	Receiver Transceiver Link
Transmitter channel	Specifies the name of the selected transmitter channel.	Transmitter Transceiver Link
TX/CMU PLL status	Provides status of whether the transmitter channel PLL is locked to the reference clock.	Transmitter Transceiver Link
Use preamble upon start	If turned on, sends the preamble word before the test pattern. If turned off, starts sending the test pattern immediately.	Transmitter Transceiver Link
VGA	(Arria 10 only) The variable gain amplifier (VGA) amplifies the signal amplitude and ensures a constant voltage swing before the data is fed to the CDR for sampling. This assignment controls the VGA output voltage swing and can be set to any value from 0 to 7.	Receiver Transceiver Link
V_{OD} control	Programmable transmitter differential output voltage.	Transmitter Transceiver Link

7.7 Scripting API Reference

You can alternatively use Tcl commands to access Transceiver Toolkit functions, rather than using the GUI. You can script various tasks, such as loading a project, creating design instances, linking device resources, and identifying high-speed serial links. You can save your project setup in a Tcl script for use in subsequent testing sessions. You can also build a custom test routine script.

After you set up and define links that describe the entire physical system, you can click **Save Tcl Script** to save the setup for future use. To run the scripts, double-click script names in the System Explorer scripts folder.

View a list of the available Tcl commands in the Tcl Console window. Select Tcl commands in the list to view descriptions, including example usage.

To view Tcl command descriptions from the Tcl Console window:

1. Type `help help`. The Console displays all Transceiver Toolkit Tcl commands.
2. Type `help` <*command name*>. The Console displays the command description.

7.7.1 Transceiver Toolkit Commands

The following tables list the available Transceiver Toolkit scripting commands.

Table 75. Transceiver Toolkit Channel_rx Commands

Command	Arguments	Function
`transceiver_channel_rx_get_data`	<*service-path*>	Returns a list of the current checker data. The results are in the order of number of bits, number of errors, and bit error rate.
`transceiver_channel_rx_get_dcgain`	<*service-path*>	Gets the DC gain value on the receiver channel.
`transceiver_channel_rx_get_dfe_tap_value`	<*service-path*> <*tap position*>	Gets the current tap value of the specified channel at the specified tap position.
`transceiver_channel_rx_get_eqctrl`	<*service-path*>	Gets the equalization control value on the receiver channel.
`transceiver_channel_rx_get_pattern`	<*service-path*>	Returns the current data checker pattern by name.
`transceiver_channel_rx_has_dfe`	<*service-path*>	Gets whether this channel has the DFE feature available.
`transceiver_channel_rx_is_checking`	<*service-path*>	Returns non-zero if the checker is running.
`transceiver_channel_rx_is_dfe_enabled`	<*service-path*>	Gets whether the DFE feature is enabled on the specified channel.
`transceiver_channel_rx_is_locked`	<*service-path*>	Returns non-zero if the checker is locked onto the incoming data.
`transceiver_channel_rx_reset_counters`	<*service-path*>	Resets the bit and error counters inside the checker.
`transceiver_channel_rx_reset`	<*service-path*>	Resets the specified channel.

continued...

Command	Arguments	Function
`transceiver_channel_rx_set_dcgain`	*<service-path> <value>*	Sets the DC gain value on the receiver channel.
`transceiver_channel_rx_set_dfe_enabled`	*<service-path><disable(0)/ enable(1)>*	Enables or disables the DFE feature on the specified channel.
`transceiver_channel_rx_set_dfe_tap_value`	*<service-path> <tap position> <tap value>*	Sets the current tap value of the specified channel at the specified tap position to the specified value.
`transceiver_channel_rx_set_dfe_adaptive`	*<service-path>*	Sets the mode of DFE adaptation. 0=off, 1=adaptive, 2= one-time adaptive
`transceiver_channel_rx_set_eqctrl`	*<service-path> <value>*	Sets the equalization control value on the receiver channel.
`transceiver_channel_rx_start_checking`	*<service-path>*	Starts the checker.
`transceiver_channel_rx_stop_checking`	*<service-path>*	Stops the checker.
`transceiver_channel_rx_set_pattern`	*<service-path> <pattern-name>*	Sets the expected pattern to the one specified by the pattern name.
`transceiver_channel_rx_set_word_aligner_enabled`	*<service-path><disable(0)/ enable(1)>*	Enables or disables the word aligner of the specified channel.
`transceiver_channel_rx_is_word_aligner_enabled`	*<service-path><disable(0)/ enable(1)>*	Gets whether the word aligner feature is enabled on the specified channel.
`transceiver_channel_rx_is_locked`	*<service-path>*	Returns non-zero if the checker is locked onto the incoming signal.
`transceiver_channel_rx_is_rx_locked_to_data`	*<service-path>*	Returns 1 if transceiver is in lock to data (LTD) mode. Otherwise 0.
`transceiver_channel_rx_is_rx_locked_to_ref`	*<service-path>*	Returns 1 if transceiver is in lock to reference (LTR) mode. Otherwise 0.

Table 76. Transceiver Toolkit Channel _tx Commands

Command	Arguments	Function
`transceiver_channel_tx_disable_preamble`	*<service-path>*	Disables the preamble mode at the beginning of generation.
`transceiver_channel_tx_enable_preamble`	*<service-path>*	Enables the preamble mode at the beginning of generation.
`transceiver_channel_tx_get_number_of_preamble_beats`	*<service-path>*	Returns the currently set number of beats to send out the preamble word.
`transceiver_channel_tx_get_pattern`	*<service-path>*	Returns the currently set pattern.
`transceiver_channel_tx_get_preamble_word`	*<service-path>*	Returns the currently set preamble word.
`transceiver_channel_tx_get_preemph0t`	*<service-path>*	Gets the pre-emphasis pre-tap value on the transmitter channel.

continued...

Command	Arguments	Function
`transceiver_channel_tx_get_preemph1t`	*<service-path>*	Gets the pre-emphasis first post-tap value on the transmitter channel.
`transceiver_channel_tx_get_preemph2t`	*<service-path>*	Gets the pre-emphasis second post-tap value on the transmitter channel.
`transceiver_channel_tx_get_vodctrl`	*<service-path>*	Gets the V_{OD} control value on the transmitter channel.
`transceiver_channel_tx_inject_error`	*<service-path>*	Injects a 1-bit error into the generator's output.
`transceiver_channel_tx_is_generating`	*<service-path>*	Returns non-zero if the generator is running.
`transceiver_channel_tx_is_preamble_enabled`	*<service-path>*	Returns non-zero if preamble mode is enabled.
`transceiver_channel_tx_set_number_of_preamble_beats`	*<service-path><number-of-preamble-beats>*	Sets the number of beats to send out the preamble word.
`transceiver_channel_tx_set_pattern`	*<service-path><pattern-name>*	Sets the output pattern to the one specified by the pattern name.
`transceiver_channel_tx_set_preamble_word`	*<service-path><preamble-word>*	Sets the preamble word to be sent out.
`transceiver_channel_tx_set_preemph0t`	*<service-path><preemph0t value>*	Sets the pre-emphasis pre-tap value on the transmitter channel.
`transceiver_channel_tx_set_preemph1t`	*<service-path><preemph1t value>*	Sets the pre-emphasis first post-tap value on the transmitter channel.
`transceiver_channel_tx_set_preemph2t`	*<service-path><preemph2t value>*	Sets the pre-emphasis second post-tap value on the transmitter channel.
`transceiver_channel_tx_set_vodctrl`	*<service-path><vodctrl value>*	Sets the V_{OD} control value on the transmitter channel.
`transceiver_channel_tx_start_generation`	*<service-path>*	Starts the generator.
`transceiver_channel_tx_stop_generation`	*<service-path>*	Stops the generator.

Table 77. Transceiver Toolkit Transceiver Toolkit Debug_Link Commands

Command	Arguments	Function
`transceiver_debug_link_get_pattern`	*<service-path>*	Gets the currently set pattern the link uses to run the test.
`transceiver_debug_link_is_running`	*<service-path>*	Returns non-zero if the test is running on the link.
`transceiver_debug_link_set_pattern`	*<service-path> <data pattern>*	Sets the pattern the link uses to run the test.
`transceiver_debug_link_start_running`	*<service-path>*	Starts running a test with the currently selected test pattern.
`transceiver_debug_link_stop_running`	*<service-path>*	Stops running the test.

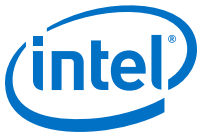


Table 78. Transceiver Toolkit Reconfig_Analog Commands

Command	Arguments	Function
`transceiver_reconfig_analog_get_logical_channel_address`	*<service-path>*	Gets the transceiver logical channel address currently set.
`transceiver_reconfig_analog_get_rx_dcgain`	*<service-path>*	Gets the DC gain value on the receiver channel specified by the current logical channel address.
`transceiver_reconfig_analog_get_rx_eqctrl`	*<service-path>*	Gets the equalization control value on the receiver channel specified by the current logical channel address.
`transceiver_reconfig_analog_get_tx_preemph0t`	*<service-path>*	Gets the pre-emphasis pre-tap value on the transmitter channel specified by the current logical channel address.
`transceiver_reconfig_analog_get_tx_preemph1t`	*<service-path>*	Gets the pre-emphasis first post-tap value on the transmitter channel specified by the current logical channel address.
`transceiver_reconfig_analog_get_tx_preemph2t`	*<service-path>*	Gets the pre-emphasis second post-tap value on the transmitter channel specified by the current logical channel address.
`transceiver_reconfig_analog_get_tx_vodctrl`	*<service-path>*	Gets the V_{OD} control value on the transmitter channel specified by the current logical channel address.
`transceiver_reconfig_analog_set_logical_channel_address`	*<service-path><logical channel address>*	Sets the transceiver logical channel address.
`transceiver_reconfig_analog_set_rx_dcgain`	*<service-path><dc_gain value>*	Sets the DC gain value on the receiver channel specified by the current logical channel address
`transceiver_reconfig_analog_set_rx_eqctrl`	*<service-path> <eqctrl value>*	Sets the equalization control value on the receiver channel specified by the current logical channel address.
`transceiver_reconfig_analog_set_tx_preemph0t`	*<service-path><preemph0t value>*	Sets the pre-emphasis pre-tap value on the transmitter channel specified by the current logical channel address.

continued...

Command	Arguments	Function
`transceiver_reconfig_analog_set_tx_preemph1t`	*<service-path><preemph1t value>*	Sets the pre-emphasis first post-tap value on the transmitter channel specified by the current logical channel address.
`transceiver_reconfig_analog_set_tx_preemph2t`	*<service-path> <preemph2t value>*	Sets the pre-emphasis second post-tap value on the transmitter channel specified by the current logical channel address.
`transceiver_reconfig_analog_set_tx_vodctrl`	*<service-path> <vodctrl value>*	Sets the V_{OD} control value on the transmitter channel specified by the current logical channel address.

.

Table 79. Transceiver Toolkit Decision Feedback Equalization (DFE) Commands

Command	Arguments	Function
`alt_xcvr_reconfig_dfe_get_logical_channel_address`	*<service-path>*	Gets the logical channel address that other `alt_xcvr_reconfig_dfe` commands use to apply.
`alt_xcvr_reconfig_dfe_is_enabled`	*<service-path>*	Gets whether the DFE feature is enabled on the previously specified channel.
`alt_xcvr_reconfig_dfe_set_enabled`	*<service-path> <disable(0)/ enable(1)>*	Enables or disables the DFE feature on the previously specified channel.
`alt_xcvr_reconfig_dfe_set_tap_value`	*<service-path> <tap position> <tap value>*	Sets the tap value at the previously specified channel at specified tap position and value.

Table 80. Channel Type Commands

Command	Arguments	Function
`get_channel_type`	*<service-path><logical-channel-num>*	Reports the detected type (GX/GT) of channel *<logical-channel-num >* for the reconfiguration block located at *<service-path>*.
`set_channel_type`	*<service-path><logical-channel-num> <channel-type>*	Overrides the detected channel type of channel *<logical-channel-num >* for the reconfiguration block located at *<service-path>* to the type specified (0:GX, 1:GT).

Table 81. Loopback Commands

Command	Arguments	Function
`loopback_get`	*<service-path>*	Returns the value of a setting or result on the loopback channel. Available results include: • Status—running or stopped. • Bytes—number of bytes sent through the loopback channel. • Errors—number of errors reported by the loopback channel. • Seconds—number of seconds since the loopback channel was started.
`loopback_set`	*<service-path>*	Sets the value of a setting controlling the loopback channel. Some settings are only supported by particular channel types. Available settings include: • Timer—number of seconds for the test run. • Size—size of the test data. • Mode—mode of the test.
`loopback_start`	*<service-path>*	Starts sending data through the loopback channel.
`loopback_stop`	*<service-path>*	Stops sending data through the loopback channel.

7.7.2 Data Pattern Generator Commands

You can use Data Pattern Generator commands to control data patterns for debugging transceiver channels. You must instantiate the Data Pattern Generator component to support these commands.

Table 82. Soft Data Pattern Generator Commands

Command	Arguments	Function
`data_pattern_generator_start`	*<service-path>*	Starts the data pattern generator.
`data_pattern_generator_stop`	*<service-path>*	Stops the data pattern generator.
`data_pattern_generator_is_generating`	*<service-path>*	Returns non-zero if the generator is running.
`data_pattern_generator_inject_error`	*<service-path>*	Injects a 1-bit error into the generator output.
`data_pattern_generator_set_pattern`	*<service-path>* *<pattern-name>*	Sets the output pattern specified by the *<pattern-name>*. In all, 6 patterns are available, 4 are pseudo-random binary sequences (PRBS), 1 is high frequency and 1 is low frequency. The PRBS7, PRBS15, PRBS23, PRBS31, HF (outputs high frequency, constant pattern of alternating 0s and 1s), and LF (outputs low frequency, constant pattern of 10b'1111100000 for 10-bit symbols and 8b'11110000 for 8-bit symbols) pattern names are defined. PRBS files are clear text and you can modify the PRBS files.
`data_pattern_generator_get_pattern`	*<service-path>*	Returns currently selected output pattern.
`data_pattern_generator_get_available_patterns`	*<service-path>*	Returns a list of available data patterns by name.
`data_pattern_generator_enable_preamble`	*<service-path>*	Enables the preamble mode at the beginning of generation.

continued...

Command	Arguments	Function
`data_pattern_generator_disable_preamble`	*<service-path>*	Disables the preamble mode at the beginning of generation.
`data_pattern_generator_is_preamble_enabled`	*<service-path>*	Returns a non-zero value if preamble mode is enabled.
`data_pattern_generator_set_preamble_word`	*<preamble-word>*	Sets the preamble word (could be 32-bit or 40-bit).
`data_pattern_generator_get_preamble_word`	*<service-path>*	Gets the preamble word.
`data_pattern_generator_set_preamble_beats`	*<service-path><number-of-preamble- beats>*	Sets the number of beats to send out in the preamble word.
`data_pattern_generator_get_preamble_beats`	*<service-path>*	Returns the currently set number of beats to send out in the preamble word.
`data_pattern_generator_fcnter_start`	*<service-path><max-cycles>*	Sets the max cycle count and starts the frequency counter.
`data_pattern_generator_check_status`	*<service-path>*	Queries the data pattern generator for current status. Returns a bitmap indicating the status, with bits defined as follows: [0]-enabled, [1]-bypass enabled, [2]-avalon, [3]-sink ready, [4]-source valid, and [5]-frequency counter enabled.
`data_pattern_generator_fcnter_report`	*<service-path><force-stop>*	Reports the current measured clock ratio, stopping the counting first depending on *<force-stop>.*

Table 83. Hard Data Pattern Generator Commands

Command	Arguments	Function
`hard_prbs_generator_start`	*<service-path>*	Starts the specified generator.
`hard_prbs_generator_stop`	*<service-path>*	Stops the specified generator.
`hard_prbs_generator_is_generating`	*<service-path>*	Checks the generation status. Returns `1` if generating, `0` otherwise.
`hard_prbs_generator_set_pattern`	*<service-path> <pattern>*	Sets the pattern of the specified hard PRBS generator to parameter `pattern`.
`hard_prbs_generator_get_pattern`	*<service-path>*	Returns the current pattern for a given hard PRBS generator.
`hard_prbs_generator_get_available_patterns`	*<service-path>*	Returns the available patterns for a given hard PRBS generator.

7.7.3 Data Pattern Checker Commands

You can use Data Pattern Checker commands to verify your generated data patterns. You must instantiate the Data Pattern Checker component to support these commands.

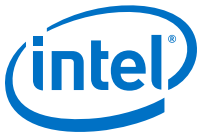


Table 84. Soft Data Pattern Checker Commands

Command	Arguments	Function
data_pattern_checker_start	<service-path>	Starts the data pattern checker.
data_pattern_checker_stop	<service-path>	Stops the data pattern checker.
data_pattern_checker_is_checking	<service-path>	Returns a non-zero value if the checker is running.
data_pattern_checker_is_locked	<service-path>	Returns non-zero if the checker is locked onto the incoming data.
data_pattern_checker_set_pattern	<service-path> <pattern-name>	Sets the expected pattern to the one specified by the <pattern-name>.
data_pattern_checker_get_pattern	<service-path>	Returns the currently selected expected pattern by name.
data_pattern_checker_get_available_patterns	<service-path>	Returns a list of available data patterns by name.
data_pattern_checker_get_data	<service-path>	Returns a list of the current checker data. The results are in the following order: number of bits, number of errors, and bit error rate.
data_pattern_checker_reset_counters	<service-path>	Resets the bit and error counters inside the checker.
data_pattern_checker_fcnter_start	<service-path><max-cycles>	Sets the max cycle count and starts the frequency counter.
data_pattern_checker_check_status	<service-path> <service-path>	Queries the data pattern checker for current status. Returns a bitmap indicating status, with bits defined as follows: [0]-enabled, [1]-locked, [2]-bypass enabled, [3]-avalon, [4]-sink ready, [5]-source valid, and [6]-frequency counter enabled.
data_pattern_checker_fcnter_report	<service-path><force-stop>	Reports the current measured clock ratio, stopping the counting first depending on <force-stop>.

Table 85. Hard Data Pattern Checker Commands

Command	Arguments	Function
hard_prbs_checker_start	<service-path>	Starts the specified hard PRBS checker.
hard_prbs_checker_stop	<service-path>	Stops the specified hard PRBS checker.
hard_prbs_checker_is_checking	<service-path>	Checks the running status of the specified hard PRBS checker. Returns a non-zero value if the checker is running.
hard_prbs_checker_set_pattern	<service-path> <pattern>	Sets the pattern of the specified hard PRBS checker to parameter <pattern>.
hard_prbs_checker_get_pattern	<service-path>	Returns the current pattern for a given hard PRBS checker.
hard_prbs_checker_get_available_patterns	<service-path>	Returns the available patterns for a given hard PRBS checker.
hard_prbs_checker_get_data	<service-path>	Returns the current bit and error count data from the specified hard PRBS checker.
hard_prbs_checker_reset_counters	<service-path>	Resets the bit and error counts of the specified hard PRBS checker.

7.8 Document Revision History

Table 86. Document Revision History

Date	Version	Changes
2016.10.31	16.1.0	• Implemented Intel rebranding. • Removed EyeQ support for Arria 10. • Renamed *"Continuous Adaptation"* to *"Adaptation Enabled"*.
May 2015	15.0.0	• Added section about Implementation Differences Between Stratix V and Arria 10. • Added section about Recommended Flow for Arria 10 Transceiver Toolkit Design with the Quartus Prime Software. • Added section about Transceiver Toolkit Troubleshooting • Updated the following sections with information about using the Transceiver Toolkit with Arria 10 devices: — Serial Bit Comparator Mode — Arria 10 Support and Limitations — Configuring BER Tests — Configuring PRBS Signal Eye Tests — Adapting Altera Design Examples — Modifying Design Examples — Configuring Custom Traffic Signal Eye Tests — Configuring Link Optimization Tests — Configuring PMA Analog Setting Control — Running BER Tests — Toolkit GUI Setting Reference • Reworked Table: Transceiver Toolkit IP Core Configuration • Replaced Figure: EyeQ Settings and Status Showing Results of Two Test Runs with Figure: EyeQ Settings and Status Showing Results of Three Test Runs. • Added Figure: Arria 10 Altera Debug Master Endpoint Block Diagram. • Added Figure: BER Test Configuration (Arria10/ Gen 10/ 20nm) Block Diagram. • Added Figure: PRBS Signal Test Configuration (Arria 10/ 20nm) Block Diagram. • Added Figure: Custom Traffic Signal Eye Test Configuration (Arria 10/ Gen 10/ 20nm) Block Diagram. • Added Figure: PMA Analog Setting Control Configuration (Arria 10/ Gen 10/ 20nm) Block Diagram. • Added Figure: One Channel Loopback Mode (Arria 10/ 20nm) Block Diagram. • Added Figure: Four Channel Loopback Mode (Arria 10/ Gen 10/ 20nm) Block Diagram. Software Version 15.0 Limitations • Transceiver Toolkit supports EyeQ for Arria 10 designs. • Supports optional hard acceleration for EyeQ. This allows for much faster EyeQ data collection. Enable this in the Arria 10 Transceiver Native PHY IP core under the **Dynamic Configuration** tab. Turn on **Enable ODI acceleration logic**.
December, 2014	14.1.0	• Added section about Arria 10 support and limitations.
June, 2014	14.0.0	• Updated GUI changes for Channel Manager with popup menus, IP Catalog, Quartus Prime, and Qsys. • Added ADME and JTAG debug link info for Arria 10. • Added instructions to run Tcl script from command line. • Added heat map display option. • Added procedure to use internal PLL to generate reconfig_clk. • Added note stating RX CDR PLL status can toggle in LTD mode.
November, 2013	13.1.0	• Reorganization and conversion to DITA.
May, 2013	13.0.0	• Added Conduit Mode Support, Serial Bit Comparator, Required Files and Tcl command tables.
November, 2012	12.1.0	• Minor editorial updates. Added Tcl help information and removed Tcl command tables. Added 28-Gbps Transceiver support section.

continued...

Date	Version	Changes
August, 2012	12.0.1	• General reorganization and revised steps in modifying Altera example designs.
June, 2012	12.0.0	• Maintenance release for update of Transceiver Toolkit features.
November, 2011	11.1.0	• Maintenance release for update of Transceiver Toolkit features.
May, 2011	11.0.0	• Added new Tcl scenario.
December, 2010	10.1.0	• Changed to new document template. Added new 10.1 release features.
August, 2010	10.0.1	• Corrected links.
July 2010	10.0.0	• Initial release.

Related Links

Altera Documentation Archive

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

8 Design Debugging with the Signal Tap Logic Analyzer

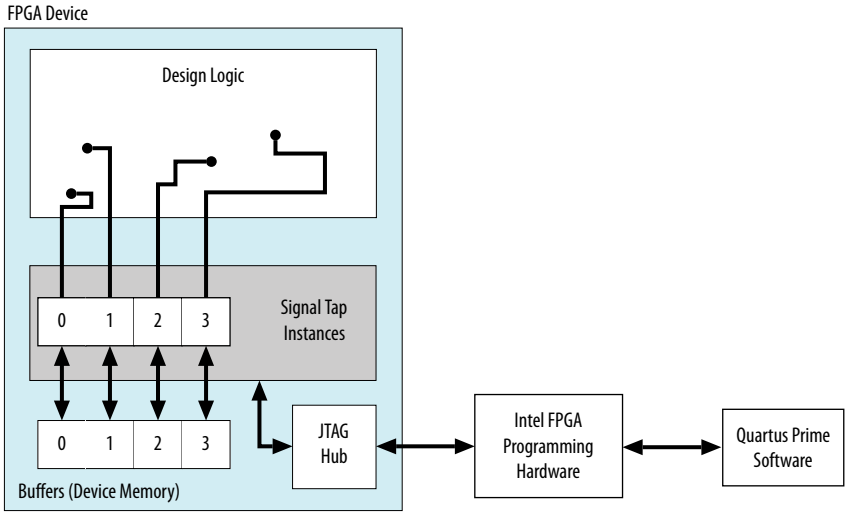
8.1 About the Signal Tap Logic Analyzer

The Signal Tap Logic Analyzer is a next-generation, system-level debugging tool that captures and displays real-time signal behavior in an FPGA design. You can examine the behavior of internal signals, without using extra I/O pins, while the design is running at full speed on an FPGA.

The Signal Tap Logic Analyzer is scalable, easy to use, and available as a stand-alone package or with a software subscription. You can debug an FPGA design by probing the state of the design's internal signals without external equipment. Define custom trigger-condition logic for greater accuracy and improved ability to isolate problems. The Signal Tap Logic Analyzer does not require external probes or design file changes to capture the state of the internal nodes or I/O pins in the design. All captured signal data is conveniently stored in device memory until you are ready to read and analyze the data.

The Signal Tap Logic Analyzer supports the highest number of channels, largest sample depth, and fastest clock speeds of any logic analyzer in the programmable logic market.

Figure 136. Signal Tap Logic Analyzer Block Diagram



This chapter is intended for any designer who wants to debug an FPGA design during normal device operation without the need for external lab equipment. Because the Signal Tap Logic Analyzer is similar to traditional external logic analyzers, familiarity with external logic analyzer operations is helpful, but not necessary.

Note: The Quartus Prime Pro Edition software uses a new methodology for settings and assignments. For example, Signal Tap assignments include only the `instance` name, not the `entity:instance` name. Refer to *Migrating to Quartus Prime Pro Edition* for more information on migrating existing Signal Tap files (`.stp`) to Quartus Prime Pro Edition.

Related Links

Migrating to Quartus Prime Pro Edition
In *Quartus Prime Pro Edition Handbook Volume 1*

8.1.1 Hardware and Software Requirements

You need the following hardware and software to perform logic analysis with the Signal Tap Logic Analyzer:

- Signal Tap Logic Analyzer software
- Download/upload cable
- Intel development kit or your design board with JTAG connection to device under test

You can use the Signal Tap Logic Analyzer that is included with the following software:

- Quartus Prime design software
- Quartus Prime Lite Edition

Alternatively, use the Signal Tap Logic Analyzer standalone software and standalone Programmer software.

The memory blocks of the device store captured data. The memory blocks transfer the data to the Quartus Prime software waveform display over a JTAG communication cable, such as or Intel FPGA Download Cable.

Table 87. Signal Tap Logic Analyzer Features and Benefits

Feature	Benefit
Quick access toolbar	Single-click operation of commonly used menu items. Hover over the icons to see tool tips.
Multiple logic analyzers in a single device	Captures data from multiple clock domains in a design at the same time.
Multiple logic analyzers in multiple devices in a single JTAG chain	Simultaneously captures data from multiple devices in a JTAG chain.
Nios II plug-in support	Easily specifies nodes, triggers, and signal mnemonics for IP, such as the Nios II processor.
Up to 10 basic, comparison, or advanced trigger conditions for each analyzer instance	Enables sending more complex data capture commands to the logic analyzer, providing greater accuracy and problem isolation.
Power-up trigger	Captures signal data for triggers that occur after device programming, but before manually starting the logic analyzer.
Custom trigger HDL object	You can code your own trigger in Verilog HDL or VHDL and tap specific instances of modules located anywhere in the hierarchy of your design without needing to manually route all the necessary connections. This simplifies the process of tapping nodes spread out across your design.

continued...

Feature	Benefit
State-based triggering flow	Enables you to organize your triggering conditions to precisely define what your logic analyzer captures.
Incremental route with rapid recompile	Manually allocate trigger input, data input, storage filter input node count, and perform a full compilation to include the Signal Tap Logic Analyzer in your design. Then, you can selectively connect, disconnect, and swap to different nodes in your design. Use Rapid Recompile to perform incremental routing and gain a 2-4x speedup over the initial full compilation.
Flexible buffer acquisition modes	The buffer acquisition control allows you to precisely control the data that is written into the acquisition buffer. Both segmented buffers and non-segmented buffers with storage qualification allow you to discard data samples that are not relevant to the debugging of your design.
MATLAB integration with included MEX function	Collects the Signal Tap Logic Analyzer captured data into a MATLAB integer matrix.
Up to 2,048 channels per logic analyzer instance	Samples many signals and wide bus structures.
Up to 128K samples in each device	Captures a large sample set for each channel.
Fast clock frequencies	Synchronous sampling of data nodes using the same clock tree driving the logic under test.
Resource usage estimator	Provides estimate of logic and memory device resources used by Signal Tap Logic Analyzer configurations.
No additional cost	The Signal Tap Logic Analyzer is included with a Quartus Prime subscription and with the Quartus Prime Lite Edition.
Compatibility with other on-chip debugging utilities	You can use the Signal Tap Logic Analyzer in tandem with any JTAG-based on-chip debugging tool, such as an In-System Memory Content editor, allowing you to change signal values in real-time while you are running an analysis with the Signal Tap Logic Analyzer.
Floating-Point Display Format	To enable, click **Edit ➤ Bus Display Format ➤ Floating-point** Supports the following: • Single-precision floating-point format **IEEE754 Single (32-bit)** • Double-precision floating-point format **IEEE754 Double (64-bit)**

Related Links

System Debugging Tools Overview on page 146

8.1.2 Open Standalone Signal Tap Logic Analyzer GUI

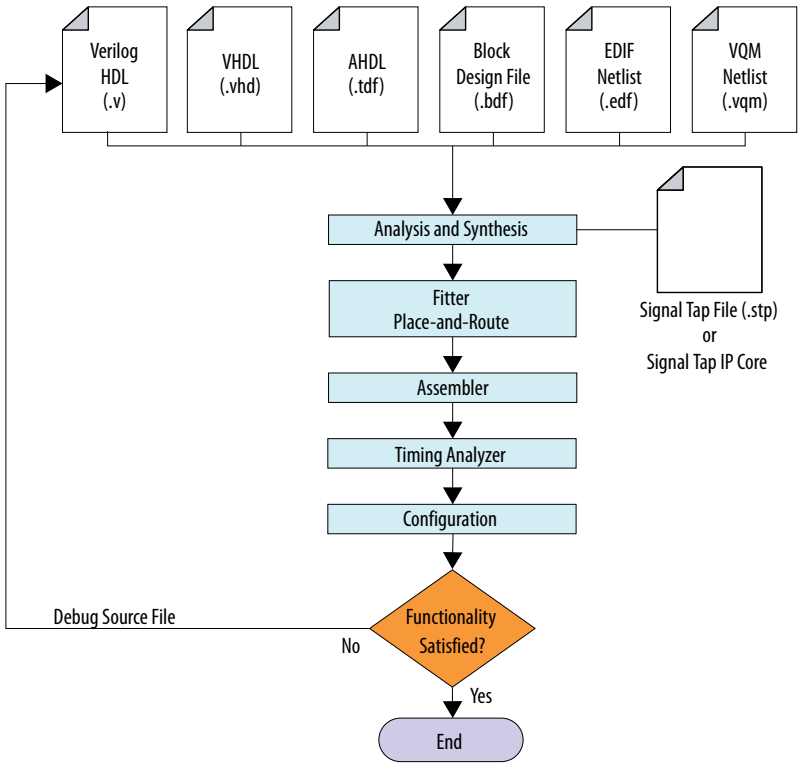
To open a new Signal Tap through the command-line, type:

```
quartus_stpw <stp_file.stp>
```

8.2 Design Flow with the Signal Tap Logic Analyzer

Add a Signal Tap file (`.stp`) and enable it in your project, or instantiate a Signal Tap IP core created with the parameter editor. The figure shows the flow of operations from initially adding the Signal Tap Logic Analyzer to your design to final device configuration, testing, and debugging.

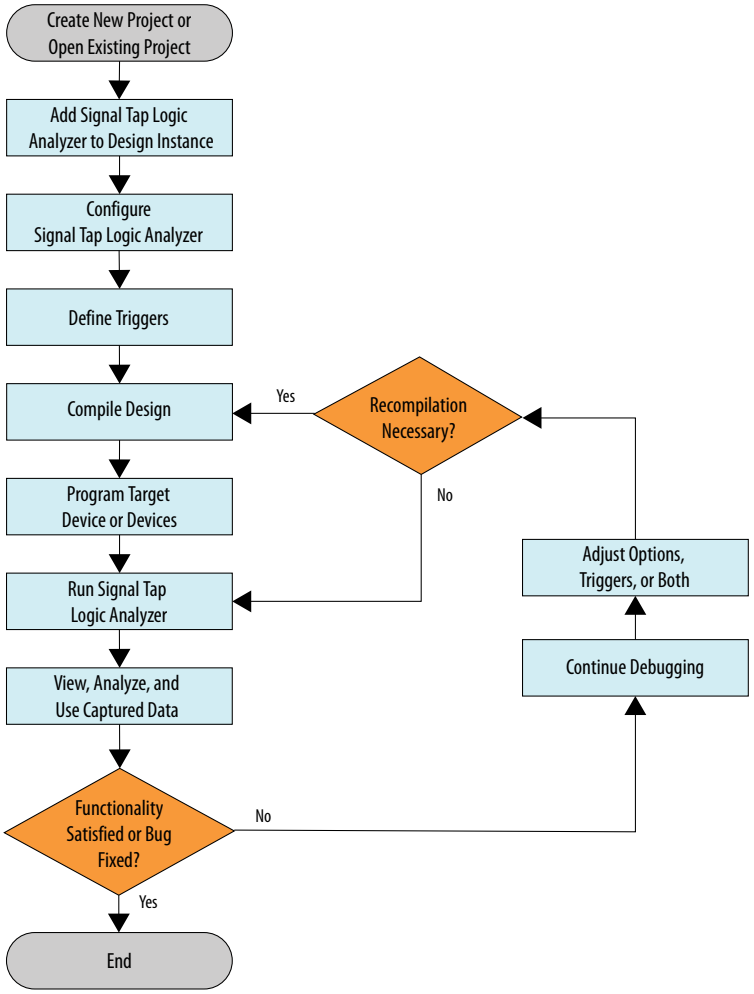
Figure 137. Signal Tap FPGA Design and Debugging Flow



8.3 Signal Tap Logic Analyzer Task Flow Overview

To use the Signal Tap Logic Analyzer to debug your design, you perform a number of tasks to add, configure, and run the logic analyzer.

Figure 138. Signal Tap Logic Analyzer Task Flow



8.3.1 Add the Signal Tap Logic Analyzer to Your Design

Create an `.stp` or create a parameterized HDL instance representation of the logic analyzer using the IP Catalog and parameter editor. If you want to monitor multiple clock domains simultaneously, add additional instances of the logic analyzer to your design, limited only by the available resources in your device.

8.3.2 Configure the Signal Tap Logic Analyzer

After you add the Signal Tap Logic Analyzer to your design, configure the logic analyzer to monitor the signals you want. Add signals manually or use a plug-in, such as the Nios II processor plug-in, to add entire sets of associated signals for a particular IP. You can also specify settings for the data capture buffer, such as its size, the method in which data is captured and stored. In devices that support memory type selection, you can specify the device memory type to use for the buffer.

Related Links

Create a Power-Up Trigger on page 303

Capture trigger events that occur during device initialization, immediately after the FPGA is powered on or reset.

8.3.3 Define Trigger Conditions

To capture and store specific signal data, set up triggers that tell the logic analyzer under what conditions to stop capturing data. The Signal Tap Logic Analyzer captures data continuously while the logic analyzer is running.

The Signal Tap Logic Analyzer allows you to define trigger conditions that range from very simple, such as the rising edge of a single signal, to very complex, involving groups of signals, extra logic, and multiple conditions. Power-Up Triggers allow you to capture data from trigger events occurring immediately after the device enters user-mode after configuration.

8.3.4 Compile the Design

Once you configure the `.stp` file and define trigger conditions, compile your project including the logic analyzer in your design.

8.3.5 Program the Target Device or Devices

When you debug a design with the Signal Tap Logic Analyzer, you can program a target device directly from the `.stp` without using the Quartus Prime Programmer. You can also program multiple devices with different designs and simultaneously debug them.

Related Links

Manage Multiple Signal Tap Files and Configurations on page 281
You can use more than one `.stp` files in one design. Use the Data Log and the SOF Manager to handle multiple Signal Tap files and configurations.

8.3.6 Run the Signal Tap Logic Analyzer

In normal device operation, you control the logic analyzer through the JTAG connection, specifying when to start looking for trigger conditions to begin capturing data. With Runtime or Power-Up Triggers, read and transfer the captured data from the on-chip buffer to the `.stp` for analysis.

8.3.7 View, Analyze, and Use Captured Data

The data you capture and read into the `.stp` file is available for analysis and debugging. You can save the data for later analysis, or convert the data to other formats for sharing and further study.

- To simplify reading and interpreting the signal data you capture, set up mnemonic tables, either manually or with a plug-in.
- To speed up debugging, use the **Locate** feature in the **Signal Tap node** list to find the locations of problem nodes in other tools in the Quartus Prime software.

8.3.8 Embed Multiple Analyzers in One FPGA

The Signal Tap Logic Analyzer Editor includes support for adding multiple logic analyzers by creating instances in the `.stp`. You can create a unique logic analyzer for each clock domain in the design.

8.3.9 Monitor FPGA Resources Used by the Signal Tap Logic Analyzer

The Signal Tap Logic Analyzer has a built-in resource estimator that calculates the logic resources and amount of memory that each logic analyzer instance uses. Furthermore, because the most demanding on-chip resource for the logic analyzer is memory usage, the resource estimator reports the ratio of total RAM usage in your design to the total amount of RAM available, given the results of the last compilation. The resource estimator provides a warning if a potential for a "no-fit" occurs.

You can see resource usage (by instance and total) in the columns of the **Instance Manager** pane of the Signal Tap Logic Analyzer Editor. Use this feature when you know that your design is running low on resources.

The logic element value that the resource usage estimator reports may vary by as much as 10% from the actual resource usage.

8.3.10 Use the Parameter Editor to Create Your Logic Analyzer

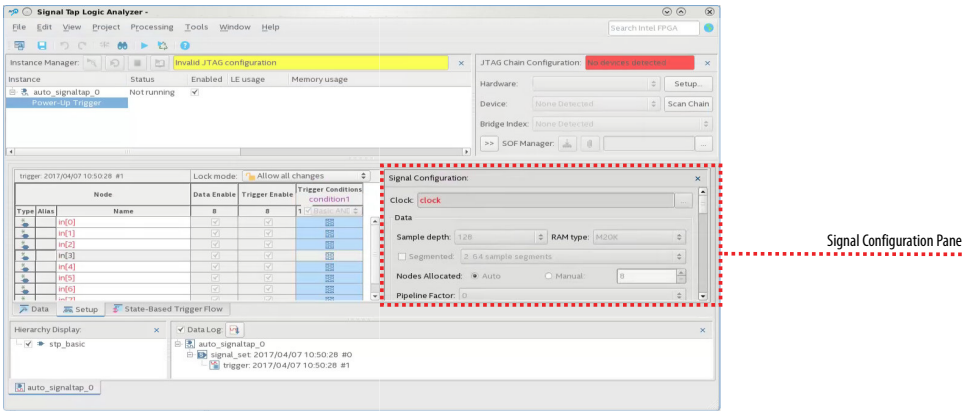
The parameter editor generates an HDL file that you instantiate in your design.

Note: The state-based trigger flow, the state machine debugging feature, and the storage qualification feature are not supported when using the parameter editor to create the logic analyzer.

8.4 Configure the Signal Tap Logic Analyzer

You can configure instances of the Signal Tap Logic Analyzer in the **Signal Configuration** pane of the **Signal Tap Logic Analyzer** window. Some settings are similar to those found on traditional external logic analyzers. Other settings are unique to the Signal Tap Logic Analyzer.

Figure 139. Signal Tap Logic Analyzer Signal Configuration Pane



Note: You can adjust fewer settings with run-time trigger conditions than with power-up trigger conditions.

8.4.1 Assign an Acquisition Clock

You assign a clock signal to control how the Signal Tap Logic Analyzer acquires data. The logic analyzer samples data on every positive (rising) edge of the acquisition clock. The logic analyzer does not support sampling on the negative (falling) edge of the acquisition clock.

You can use any signal in your design as the acquisition clock. However, for best results in data acquisition, use a global, non-gated clock that is synchronous to the signals under test. Using a gated clock as your acquisition clock can result in unexpected data that does not accurately reflect the behavior of your design. The Quartus Prime static timing analysis tools show the maximum acquisition clock frequency at which you can run your design. To find the maximum frequency of the logic analyzer clock, refer to the Timing Analysis section of the Compilation Report.

Caution: Be careful when using a recovered clock from a transceiver as an acquisition clock for the Signal Tap Logic Analyzer. A recovered clock can cause incorrect or unexpected behavior, particularly when the transceiver recovered clock is the acquisition clock with the power-up trigger feature.

If you do not assign an acquisition clock in the Signal Tap Logic Analyzer Editor, Quartus Prime software automatically creates a clock pin called `auto_stp_external_clk`. You must make a pin assignment to this pin and ensure that a clock signal in your design drives the acquisition clock.

Related Links

- Adding Signals with a Plug-In on page 268
  Instead of adding individual or grouped signals through the **Node Finder**, you can use a plug-in to add groups of relevant signals of a particular type of IP.
- Managing Device I/O Pins
  In *Quartus Prime Pro Edition Handbook Volume 2*

8.4.2 Assigning Data Signals Using the Technology Map Viewer

You can use the Technology Map Viewer to add post-fit signal names easily. To do so, launch the Technology Map Viewer (post-fitting) after compilation. When you find the desired node, copy the node to either the active `.stp` for your design or a new `.stp`.

To launch the Technology Map Viewer, click **Tools ➤ Netlist Viewers ➤ Technology Map Viewer (Post-Fitting)** in the **Quartus Prime** window.

8.4.3 Add Signals to the Signal Tap File

Add the signals that you want to monitor to the `.stp` node list. You can also select signals to define triggers. You can assign the following two signal types:

- **Pre-synthesis**—These signals exist after design elaboration, but before any synthesis optimizations are done. This set of signals must reflect your Register Transfer Level (RTL) signals.
- **Post-fitting**—These signals exist after physical synthesis optimizations and place-and-route.

Quartus Prime software does not limit the number of signals available for monitoring in the Signal Tap window waveform display. However, the number of channels available is directly proportional to the number of logic elements (LEs) or adaptive logic modules (ALMs) in the device. Therefore, there is a physical restriction on the number of channels that are available for monitoring. Signals shown in blue text are post-fit node names. Signals shown in black text are pre-synthesis node names.

Note: The Quartus Prime Pro Edition software uses only the instance name, and not the entity name, in the form of:

`a|b|c`

not `a_entity:a|b_entity:b|c_entity:c`

After successful Analysis and Elaboration, invalid signals appear in red. Unless you are certain that these signals are valid, remove them from the `.stp` file for correct operation. The Signal Tap Status Indicator also indicates if an invalid node name exists in the `.stp` file.

You can tap signals if a routing resource (row or column interconnects) exists to route the connection to the Signal Tap instance. For example, you can't tap signals that exist in the I/O element (IOE), because there are no direct routing resources from the signal in an IOE to a core logic element. For input pins, you can tap the signal that is driving a logic array block (LAB) from an IOE, or, for output pins, you can tap the signal from the LAB that is driving an IOE.

8.4.3.1 About Adding Pre-Synthesis Signals

When you add pre-synthesis signals, make all connections to the Signal Tap Logic Analyzer before synthesis. The Compiler allocates logic and routing resources to make the connection as if you changed your design files. For signals driving to and from IOEs, pre-synthesis signal names coincide with the pin's signal names.

8.4.3.2 About Adding Post-Fit Signals

In the case of post-fit signals, connections that you make to the Signal Tap Logic Analyzer are the signal names from the actual atoms in your post-fit netlist. You can only make a connection if the signals are part of the existing post-fit netlist, and existing routing resources are available from the signal of interest to the Signal Tap Logic Analyzer.

In the case of post-fit output signals, tap the `COMBOUT` or `REGOUT` signal that drives the IOE block. For post-fit input signals, signals driving into the core logic coincide with the pin's signal name.

Note: Because `NOT`-gate push back applies to any register that you tap, the signal from the atom may be inverted. You can check this by locating the signal in either the Resource Property Editor or the Technology Map Viewer. You can also use the Technology Map viewer and the Resource Property Editor to find post-fit node names.

Related Links

Design Flow with the Netlist Viewers
In *Quartus Prime Pro Edition Handbook Volume 1*

8.4.3.3 Preserving Signals

The Quartus Prime software optimizes the RTL signals during synthesis and place-and-route. RTL signal names may not appear in the post-fit netlist after optimizations. For example, the compilation process can add tildes (~) to nets that fan-out from a node, making it difficult to decipher which signal nets they actually represent.

The Quartus Prime software provides synthesis attributes that prevent the Compiler to perform any optimization on the specified signals, allowing them to persist into the post-fit netlist. However, using these attributes can increase device resource utilization or decrease timing performance.

- `keep`—Prevents removal of combinational signals during optimization.
- `preserve`—Prevents removal of registers during optimization.

If you are debugging an IP core, such as the Nios II CPU or other encrypted IP, you might need to preserve nodes from the core to keep available for debugging with the Signal Tap Logic Analyzer. Preserving nodes is often necessary when a plug-in is used to add a group of signals for a particular IP.

8.4.3.4 Node List Signal Use Options

When you add a signal to the node list, you can select options that specify how the logic analyzer uses the signal.

To prevent a signal from triggering the analysis, disable the signal's **Trigger Enable** option in the `.stp` file. This option is useful when you only want to see the signal's captured data.

You can turn off the ability to view data for a signal by disabling the **Data Enable** column in the `.stp` file. This option is useful when you want to trigger on a signal, but have no interest in viewing that signal's data.

Related Links

Define Triggers on page 283
Specify various types of trigger conditions using the Signal Tap Logic Analyzer on the **Signal Configuration** pane.

8.4.3.4.1 Disabling and Enabling a Signal Tap Instance

Disable and enable Signal Tap instances in the **Instance Manager** pane. Physically adding or removing instances requires recompilation after disabling and enabling a Signal Tap instance.

8.4.3.5 Untappable Signals

Not all of the post-fitting signals in your design are available in the **Signal Tap : post-fitting filter** in the **Node Finder** dialog box.

You cannot tap any of the following signal types:

- **Post-fit output pins**—You cannot tap a post-fit output pin directly. To make an output signal visible, tap the register or buffer that drives the output pin. This includes pins defined as bidirectional.
- **Signals that are part of a carry chain**—You cannot tap the carry out (`cout0` or `cout1`) signal of a logic element. Due to architectural restrictions, the carry out signal can only feed the carry in of another LE.
- **JTAG Signals**—You cannot tap the JTAG control (`TCK`, `TDI`, `TDO`, and `TMS`) signals.
- **ALTGXB IP core**—You cannot directly tap any ports of an ALTGXB instantiation.
- **LVDS**—You cannot tap the data output from a serializer/deserializer (SERDES) block.
- **DQ**, **DQS Signals**—You cannot directly tap the `DQ` or `DQS` signals in a DDR/DDRII design.

8.4.4 Adding Signals with a Plug-In

Instead of adding individual or grouped signals through the **Node Finder**, you can use a plug-in to add groups of relevant signals of a particular type of IP. Besides easy signal addition, plug-ins provide features such as pre-designed mnemonic tables, useful for trigger creation and data viewing, as well as the ability to disassemble code in captured data. The Signal Tap Logic Analyzer comes with one plug-in for the Nios II processor.

The Nios II plug-in, for example, creates one mnemonic table in the **Setup** tab and two tables in the **Data** tab:

- **Nios II Instruction** (**Setup** tab)—Capture all the required signals for triggering on a selected instruction address.
- **Nios II Instance Address** (**Data** tab)—Display address of executed instructions in hexadecimal format or as a programming symbol name if defined in an optional Executable and Linking Format (`.elf`) file.
- **Nios II Disassembly** (**Data** tab)—Display disassembled code from the corresponding address.

To add signals to the `.stp` file using a plug-in, perform the following steps after running Analysis and Elaboration on your design:

1. To ensure that all the required signals are available, in the Quartus Prime software, click **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Synthesis)**. Turn on **Create debugging nodes for IP cores**. All the signals included in the plug-in are added to the node list.
2. Right-click the node list. On the **Add Nodes with Plug-In** submenu, select the plug-in you want to use, such as the included plug-in named **Nios II**. The **Select Hierarchy Level** dialog box appears showing the IP hierarchy of your design. If the IP for the selected plug-in does not exist in your design, a message informs you that you cannot use the selected plug-in.
3. Select the IP that contains the signals you want to monitor with the plug-in, and click **OK**.

— If all the signals in the plug-in are available, a dialog box might appear, depending on the plug-in, where you can specify options for the plug-in.

4. With the Nios II plug-in, you can optionally select an `.elf` containing program symbols from your Nios II Integrated Development Environment (IDE) software design. Specify options for the selected plug-in, and click **OK**.

Related Links

- Define Triggers on page 283
  Specify various types of trigger conditions using the Signal Tap Logic Analyzer on the **Signal Configuration** pane.
- View, Analyze, and Use Captured Data on page 263
  The data you capture and read into the `.stp` file is available for analysis and debugging.

8.4.5 Add Finite State Machine State Encoding Registers

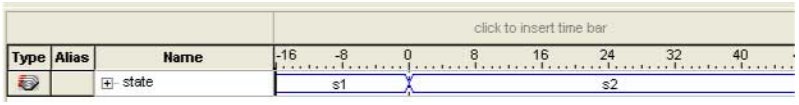
Detect FSMs in your compiled design with the Signal Tap Logic Analyzer.

Finding the signals to debug finite state machines (FSM) can be challenging. Finding nodes from the post-fit netlist may be impossible, as FSM encoding signals may be changed or optimized away during synthesis and place-and-route. If you can find all the relevant nodes in the post-fit netlist or you used the nodes from the pre-synthesis netlist, an additional step is required to find and map FSM signal values to the state names that you specified in your HDL.

The Signal Tap Logic Analyzer configuration automatically tracks the FSM state signals as well as state encoding through the compilation process. Shortcut menu commands allow you to add all the FSM state signals to your logic analyzer with a single command. For each FSM added to your Signal Tap configuration, the FSM debugging feature adds a mnemonic table to map the signal values to the state enumeration that you provided in your source code. The mnemonic tables enable you to visualize state machine transitions in the waveform viewer. The FSM debugging feature supports adding FSM signals from both the pre-synthesis and post-fit netlists.

Figure 140. Decoded FSM Mnemonics

The waveform viewer with decoded signal values from a state machine added with the FSM debugging feature.



Related Links

- Recommended HDL Coding Styles
  In *Quartus Prime Pro Edition Handbook Volume 1*
- State Machine HDL Guidelines
  In *Quartus Prime Pro Edition Handbook Volume 1*

8.4.5.1 Modify and Restore Mnemonic Tables for State Machines

Edit any mnemonic table using the **Mnemonic Table Setup** dialog box. When you add FSM state signals via the FSM debugging feature, the Signal Tap Logic Analyzer GUI creates a mnemonic table using the format <*StateSignalName*>_table, where **StateSignalName** is the name of the state signals that you have declared in your RTL.

If you want to restore a mnemonic table that was modified, right-click anywhere in the node list window and select **Recreate State Machine Mnemonics**. By default, restoring a mnemonic table overwrites the existing mnemonic table that you modified. To restore a FSM mnemonic table to a new record, turn off **Overwrite existing mnemonic table** in the **Recreate State Machine Mnemonics** dialog box.

Note: If you have added or deleted a signal from the FSM state signal group from within the setup tab, delete the modified register group and add the FSM signals back again.

Related Links

Creating Mnemonics for Bit Patterns on page 317
The mnemonic table feature allows you to assign a meaningful name to a set of bit patterns, such as a bus.

8.4.5.2 Additional Considerations for State Machines in Signal Tap

- The Signal Tap configuration GUI recognizes state machines from your design only if you use Quartus Prime Synthesis. Conversely, the state machine debugging feature is not able to track the FSM signals or state encoding if you use other EDA synthesis tools.
- If you add post-fit FSM signals, the Signal Tap Logic Analyzer FSM debug feature may not track all optimization changes that are a part of the compilation process.
- If the following two specific optimizations are enabled, the Signal Tap FSM debug feature may not list mnemonic tables for state machines in the design:
  - If you enabled the **Physical Synthesis** optimization, state registers may be resource balanced (register retiming) to improve f_{MAX}. The FSM debug feature does not list post-fit FSM state registers if register retiming occurs.
  - The FSM debugging feature does not list state signals that the Compiler packed into RAM and DSP blocks during synthesis or Fitter optimizations.
- You can still use the FSM debugging feature to add pre-synthesis state signals.

Related Links

Enabling Physical Synthesis Optimization
In *Quartus Prime Pro Edition Handbook Volume 1*

8.4.6 Specify the Sample Depth

The **Sample depth** setting specifies the number of samples the Signal Tap Logic Analyzer captures and stores, for each signal in the captured data buffer. To specify the sample depth, select the desired number in the **Sample Depth** drop-down menu. The sample depth ranges from 0 to 128K.

If device memory resources are limited, you may not be able to successfully compile your design with the sample buffer size you have selected. Try reducing the sample depth to reduce resource usage.

Related Links

Signal Configuration Pane (View Menu) (Signal Tap Logic Analyzer)
in Quartus Prime Help

8.4.7 Capture Data to a Specific RAM Type

You have the option to select the RAM type where the Signal Tap Logic Analyzer stores acquisition data. Once you allocate the Signal Tap Logic Analyzer buffer to a particular RAM block, the entire RAM block becomes a dedicated resource for the logic analyzer.

RAM selection allows you to preserve a specific memory block for your design, and allocate another portion of memory for Signal Tap Logic Analyzer data acquisition.

To specify the RAM type to use for the Signal Tap Logic Analyzer buffer, go to the **Signal Configuration** pane in the **Signal Tap** window, and select one **Ram type** from the drop-down menu.

Use this feature only when the acquired data is smaller than the available memory of the RAM type that you selected. The amount of data appears in the Signal Tap resource estimator.

Related Links

Signal Configuration Pane (View Menu) (Signal Tap Logic Analyzer)
in Quartus Prime Help

8.4.8 Select the Buffer Acquisition Mode

When you specify how the logic analyzer organizes the captured data buffer, you can potentially reduce the amount of memory that Signal Tap requires for data acquisition.

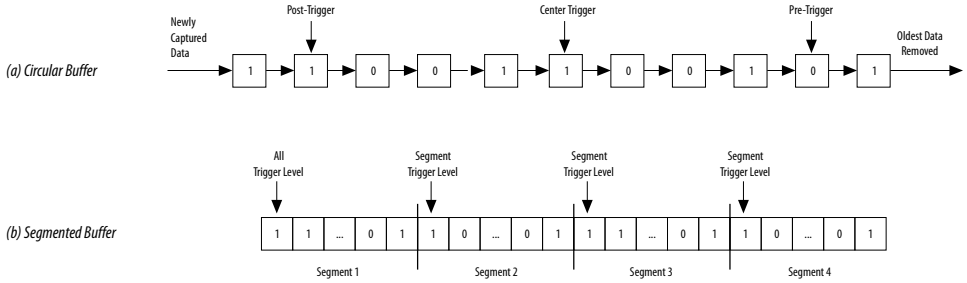
There are two types of acquisition buffer within the Signal Tap Logic Analyzer—a non-segmented (or circular) buffer and a segmented buffer.

- With a non-segmented buffer, the Signal Tap Logic Analyzer treats entire memory space as a single FIFO, continuously filling the buffer until the logic analyzer reaches a defined set of trigger conditions.
- With a segmented buffer, the memory space is split into separate buffers. Each buffer acts as a separate FIFO with its own set of trigger conditions, and behaves as a non-segmented buffer. Only a single buffer is active during an acquisition. The Signal Tap Logic Analyzer advances to the next segment after the trigger condition or conditions for the active segment has been reached.

When using a non-segmented buffer, you can use the storage qualification feature to determine which samples are written into the acquisition buffer. Both the segmented buffers and the non-segmented buffer with the storage qualification feature help you maximize the use of the available memory space.

Figure 141. Buffer Type Comparison in the Signal Tap Logic Analyzer

The figure illustrates the differences between the two buffer types.



Both non-segmented and segmented buffers can use a preset trigger position (Pre-Trigger, Center Trigger, Post-Trigger). Alternatively, you can define a custom trigger position using the **State-Based Triggering** tab. Refer to *Specify Trigger Position* for more details.

Notes to figure:

Related Links

- Specify Trigger Position on page 302
  Specify the amount of data acquired before and after a trigger event.
- Using the Storage Qualifier Feature on page 274
  The Storage Qualifier feature allows you to filter out individual samples not relevant to debugging the design.

8.4.8.1 Non-Segmented Buffer

The non-segmented buffer is the default buffer type in the Signal Tap Logic Analyzer.

At runtime, the logic analyzer stores data in the buffer until the buffer fills up. From that point on, new data overwrites the oldest data, until a specific trigger event occurs. The amount of data the buffer captures after the trigger event depends on the **Trigger position** setting:

- To capture most data before the trigger occurs, select **Post trigger position** from the list
- To capture most data after the trigger, select **Pre trigger position**.
- To center the trigger position in the data, select **Center trigger position**.

Alternatively, use the custom State-based triggering flow to define a custom trigger position within the capture buffer.

Related Links

Specify Trigger Position on page 302
Specify the amount of data acquired before and after a trigger event.

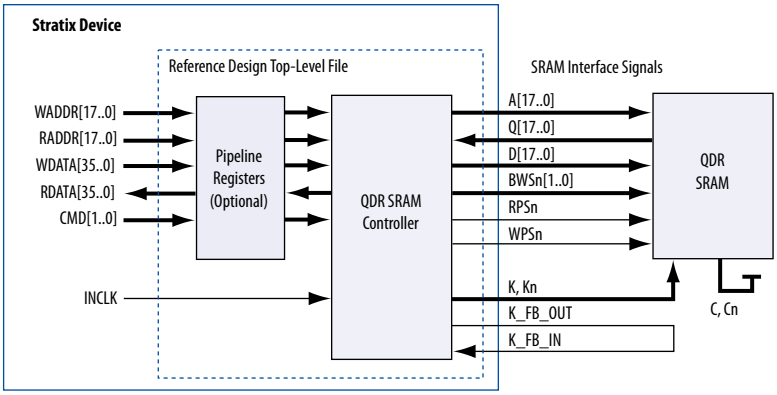
8.4.8.2 Segmented Buffer

In a segmented buffer, the acquisition memory is split into evenly sized segments, with a set of trigger conditions defined for each segment. Each segment acts as a non-segmented buffer. A segmented buffer allows you to debug systems that contain relatively infrequent recurring events.

If you want to have separate trigger conditions for each of the segmented buffers, you must use the state-based trigger flow. The figure shows an example of a segmented buffer system.

Figure 142. System that Generates Recurring Events

In this design, you want to ensure that the correct data is written to the SRAM controller by monitoring the RDATA port whenever the address `H'0F0F0F0F` is sent into the RADDR port.



With the buffer acquisition feature. you can monitor multiple read transactions from the SRAM device without running the Signal Tap Logic Analyzer again, because you split the memory to capture the same event multiple times, without wasting allocated memory. The buffer captures as many cycles as the number of segments you define under the **Data** settings.

To enable and configure buffer acquisition, select **Segmented** in the Signal Tap Logic Analyzer Editor and determine the number of segments to use. In the example in the figure, selecting sixty-four 64-sample segments allows you to capture 64 read cycles.

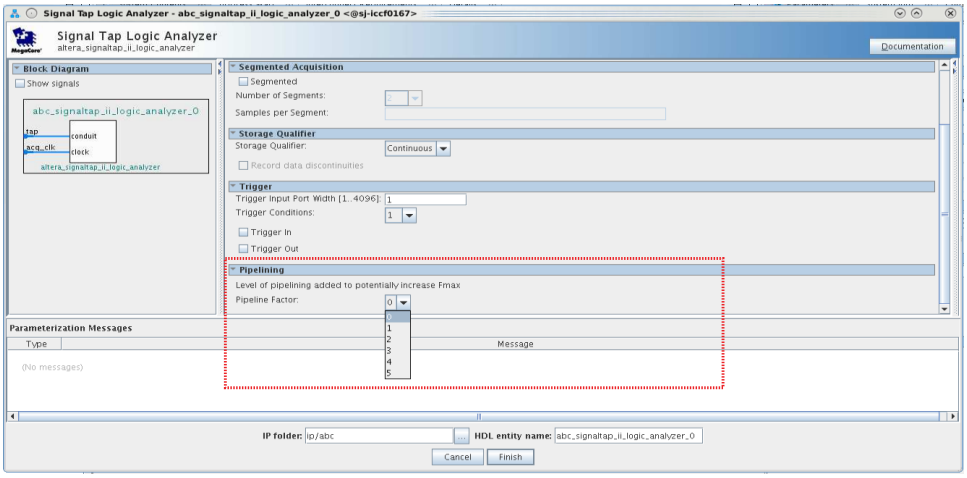
8.4.9 Specify the Pipeline Factor

The **Pipeline factor** setting indicates the number of pipeline registers that you can add to boost the f_{MAX} of the Signal Tap Logic Analyzer. You can specify the pipeline factor in the **Signal Configuration** pane. The pipeline factor ranges from 0 to 5, with a default value of 0.

You can also set the pipeline factor when you instantiate the Signal Tap Logic Analyzer component from your Qsys Pro system:

1. Double-click **Signal Tap Logic Analyzer** component in the IP Catalog.
2. Specify the **Pipeline Factor**, along with other parameter values.

Figure 143. Specifying the Pipeline Factor from Qsys Pro



Note: Setting the pipeline factor does not guarantee an increase in f_{MAX}, as the pipeline registers may not be in the critical paths.

8.4.10 Using the Storage Qualifier Feature

The Storage Qualifier feature allows you to filter out individual samples not relevant to debugging the design.

The Signal Tap Logic Analyzer offers a snapshot in time of the data stored in the acquisition buffers. By default, the Signal Tap Logic Analyzer writes into acquisition memory with data samples on every clock cycle. With a non-segmented buffer, there is one data window that represents a comprehensive snapshot of the datastream. Conversely, segmented buffers use several smaller sampling windows spread out over more time, with each sampling window representing a contiguous data set.

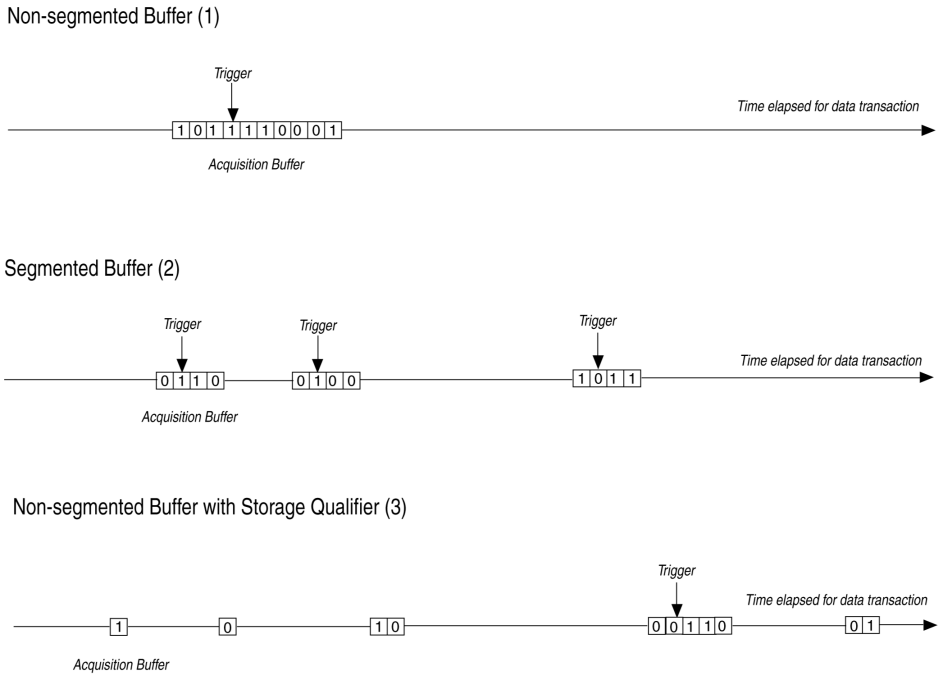
With analysis using acquisition buffers you can capture most functional errors in a chosen signal set, provided adequate trigger conditions and a generous sample depth for the acquisition. However, each data window can have a considerable amount of unnecessary data; for example, long periods of idle signals between data bursts. The default behavior in the Signal Tap Logic Analyzer doesn't discard the redundant sample bits.

The Storage Qualifier feature allows you to establish a condition that acts as a write enable to the buffer during each clock cycle of data acquisition, thus allowing a more efficient use of acquisition memory over a longer period of analysis.

Because you can create a discontinuity between any two samples in the buffer, the Storage Qualifier feature is equivalent to creating a custom segmented buffer in which the number and size of segment boundaries are adjustable.

Note: You can only use the Storage Qualifier feature with a non-segmented buffer. The IP Catalog flow only supports the Input Port mode for the Storage Qualifier feature.

Figure 144. Data Acquisition Using Different Modes of Controlling the Acquisition Buffer



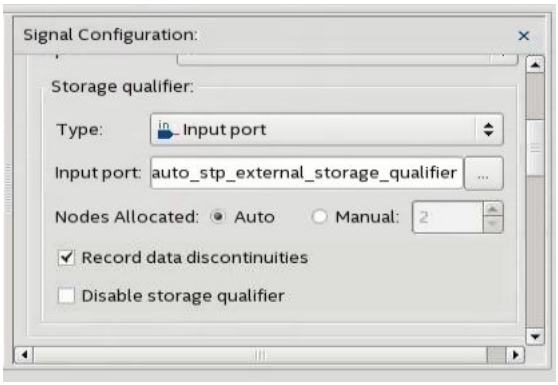
Notes to figure:

1. Non-segmented buffers capture a fixed sample window of contiguous data.
2. Segmented buffers divide the buffer into fixed sized segments, with each segment having an equal sample depth.
3. Storage Qualifier allows you to define a custom sampling window for each segment you create with a qualifying condition, thus potentially allowing a larger time scale of coverage.

There are six storage qualifier types available under the Storage Qualifier feature:

- **Continuous** (default) Turns the Storage Qualifier off.
- **Input port**
- **Transitional**
- **Conditional**
- **Start/Stop**
- **State-based**

Figure 145. Storage Qualifier Settings



Upon the start of an acquisition, the Signal Tap Logic Analyzer examines each clock cycle and writes the data into the buffer based upon the storage qualifier type and condition. Acquisition stops when a defined set of trigger conditions occur.

The Signal Tap Logic Analyzer evaluates trigger conditions independently of storage qualifier conditions.

Related Links

Define Trigger Conditions on page 263
To capture and store specific signal data, set up triggers that tell the logic analyzer under what conditions to stop capturing data.

8.4.10.1 Input Port Mode

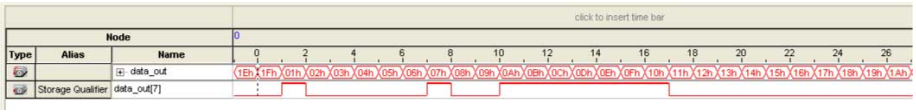
When using the Input port mode, the Signal Tap Logic Analyzer takes any signal from your design as an input. During acquisition, if the signal is high on the clock edge, the Signal Tap Logic Analyzer stores the data in the buffer. If the signal is low on the clock edge, the Logic Analyzer ignores the data sample. If you don't specify an internal node, the Logic Analyzer creates and connects a pin to this input port.

If you are creating a Signal Tap Logic Analyzer instance through an `.stp` file, specify the storage qualifier signal using the input port field located on the **Setup** tab. You must specify this port for your project to compile.

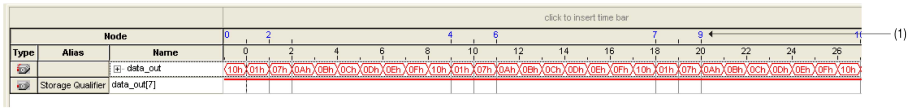
If you use the parameter editor, the storage qualification input port, if specified, appears in the generated instantiation template. You can then connect this port to a signal in your RTL.

Figure 146. Comparing Continuous and Input Port Capture Mode in Data Acquisition of a Recurring Data Pattern

- Continuous Mode:



- Input Port Storage Qualifier:



(1) Markers display samples when the logic analyzer paused a write into acquisition memory. These markers are enabled with the option "Record data discontinuities."

8.4.10.2 Transitional Mode

In Transitional mode, the Logic Analyzer monitors changes in a set of signals, and writes new data in the acquisition buffer only when it detects a change. You select the signals for monitoring using the check boxes in the **Storage Qualifier** column.

Figure 147. Transitional Storage Qualifier Setup

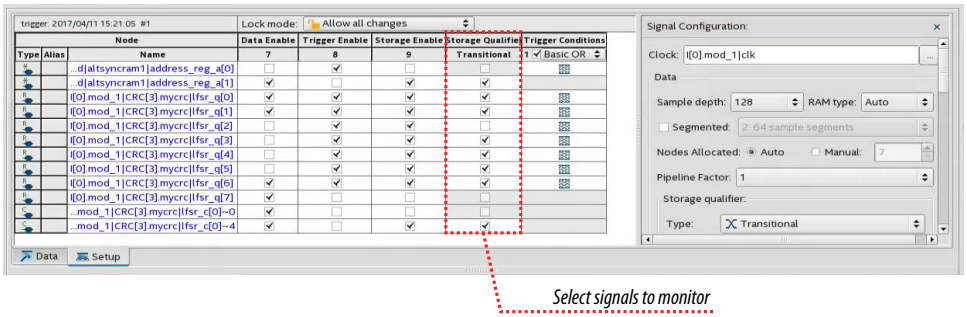
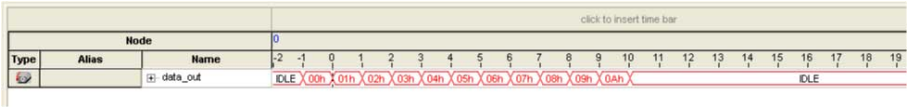
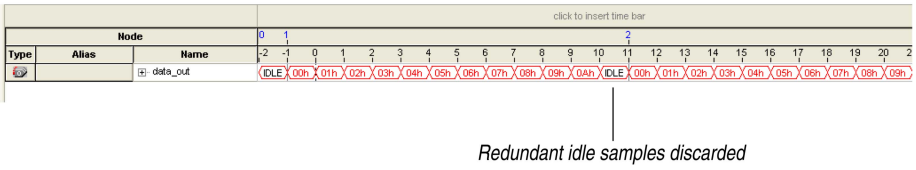


Figure 148. Comparing Continuous and Transitional Capture Mode in Data Acquisition of a Recurring Data Pattern

- Continuous:



- Transitional mode:



8.4.10.3 Conditional Mode

In Conditional mode, the Signal Tap Logic Analyzer determines whether to store a sample by evaluating a combinational function of predefined signals within the node list. The Signal Tap Logic Analyzer writes into the buffer during the clock cycles in which the condition you specify evaluates TRUE.

You can select either **Basic AND**, **Basic OR**, **Comparison**, or **Advanced** storage qualifier conditions. A **Basic AND** or **Basic OR** condition matches each signal to one of the following:

- **Don't Care**
- **Low**
- **High**
- **Falling Edge**
- **Rising Edge**
- **Either Edge**

If you specify a **Basic AND** storage qualifier condition for more than one signal, the Signal Tap Logic Analyzer evaluates the logical AND of the conditions.

You can specify any other combinational or relational operators with the enabled signal set for storage qualification through advanced storage conditions.

You can define storage qualification conditions similar to the manner in which you define trigger conditions.

Figure 149. Conditional Storage Qualifier Setup

The figure details the conditional storage qualifier setup in the `.stp` file.

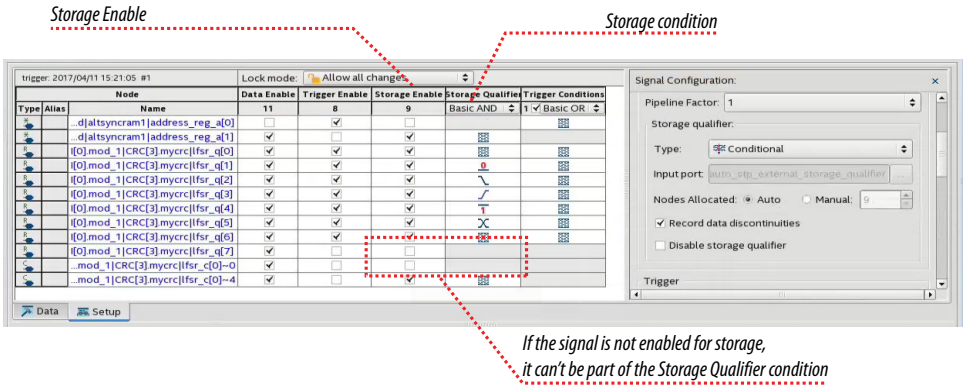
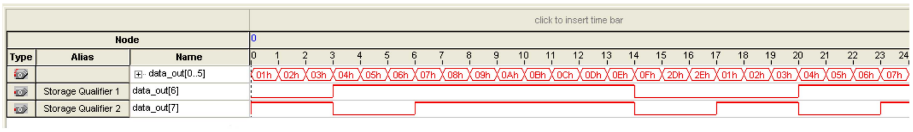


Figure 150. Comparing Continuous and Conditional Capture Mode in Data Acquisition of a Recurring Data Pattern

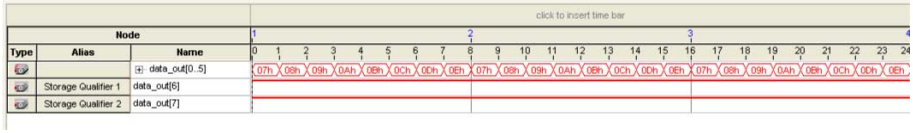
The data pattern is the same in both cases.

- Continuous sampling capture mode:



(1) Storage Qualifier condition is set up to evaluate data_out[6] AND data_out[7].

- Conditional sampling capture mode:



Related Links

- Basic Trigger Conditions on page 283
  If you select the **Basic AND** or **Basic OR** trigger type, you must specify the trigger pattern for each signal you have added in the `.stp`.
- Comparison Trigger Conditions on page 284
  The **Comparison** trigger allows you to compare multiple grouped bits of a bus to an expected integer value by specifying simple comparison conditions on the bus node.
- Advanced Trigger Conditions on page 286
  To capture data for a given combination of conditions, build an advanced trigger.

8.4.10.4 Start/Stop Mode

The Start/Stop mode uses two sets of conditions, one to start data capture and one to stop data capture. If the start condition evaluates to TRUE, Signal Tap Logic Analyzer stores the buffer data every clock cycle until the stop condition evaluates to TRUE, which then pauses the data capture. The Logic Analyzer ignores additional start signals received after the data capture starts. If both start and stop evaluate to TRUE at the same time, the Logic Analyzer captures a single cycle.

Note: You can force a trigger by pressing the **Stop** button if the buffer fails to fill to completion due to a stop condition.

Figure 151. Start/Stop Mode Storage Qualifier Setup

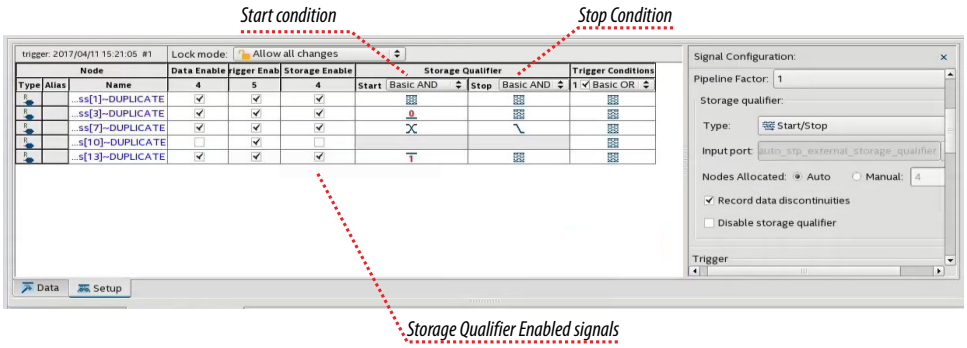
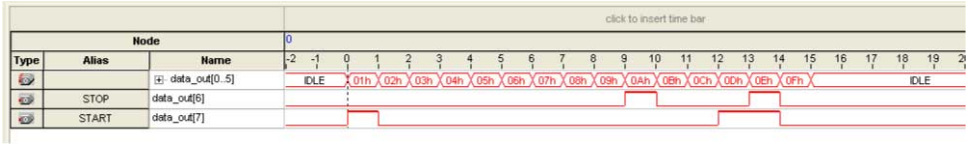
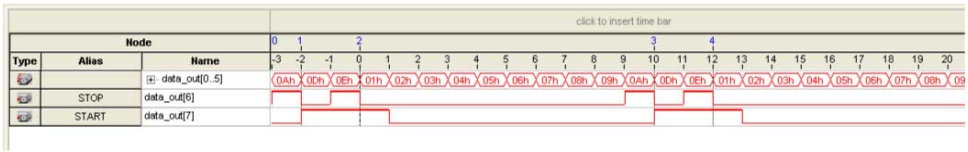


Figure 152. Comparing Continuous and Start/Stop Acquisition Modes for a Recurring Data Pattern

- Continuous Mode:



- Start/Stop Storage Qualifier:



8.4.10.5 State-Based

The State-based storage qualification mode is part of the State-based triggering flow. The state based triggering flow evaluates a conditional language to define how the Signal Tap Logic Analyzer writes data into the buffer. With the State-based trigger flow, you have command over boolean and relational operators to guide the execution flow for the target acquisition buffer.

When you enable the storage qualifier feature for the State-based flow, two additional commands become available: `start_store` and `stop_store`. These commands are similar to the Start/Stop capture conditions. Upon the start of acquisition, the Signal Tap Logic Analyzer doesn't write data into the buffer until a `start_store` action is performed. The `stop_store` command pauses the acquisition. If both `start_store` and `stop_store` actions occur within the same clock cycle, the Logic Analyzer stores a single sample into the acquisition buffer.

Related Links

State-Based Triggering on page 293

Custom State-based triggering provides the most control over triggering condition arrangement.

8.4.10.6 Showing Data Discontinuities

When you turn on **Record data discontinuities**, the Signal Tap Logic Analyzer marks the samples during which the acquisition paused from a storage qualifier. This marker is displayed in the waveform viewer after acquisition completes.

8.4.10.7 Disable Storage Qualifier

You can turn off the storage qualifier quickly with the **Disable Storage Qualifier** option, and perform a continuous capture. This option is run-time reconfigurable; that is, the setting can be changed without recompiling the project. Changing storage qualifier mode from the **Type** field requires a recompilation of the project.

Related Links

Runtime Reconfigurable Options on page 311

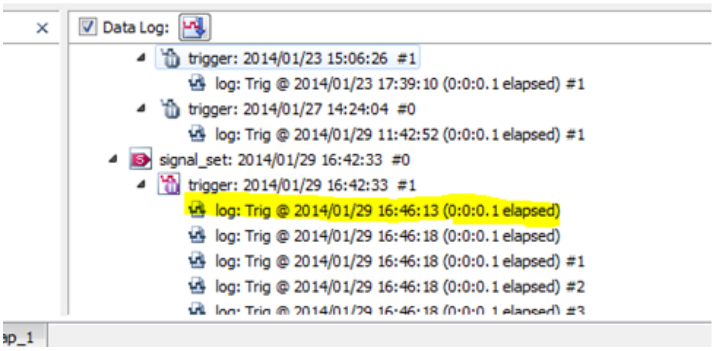
8.4.11 Manage Multiple Signal Tap Files and Configurations

You can use more than one `.stp` files in one design. Use the Data Log and the SOF Manager to handle multiple Signal Tap files and configurations. Each file potentially has a different group of monitored signals. These signal groups make it possible to debug diverse blocks in your design. In turn, you can use each group of signals to define specific sets of trigger conditions.

To function correctly, the settings in the `.stp` file you use at runtime must match the Signal Tap settings in the `.sof` file you use to program the device.

Figure 153. Data Log

The Data Log displays and stores the multiple Signal Tap configurations within a single `.stp` file. The figure shows two signal set configurations with multiple trigger conditions in one `.stp` file.

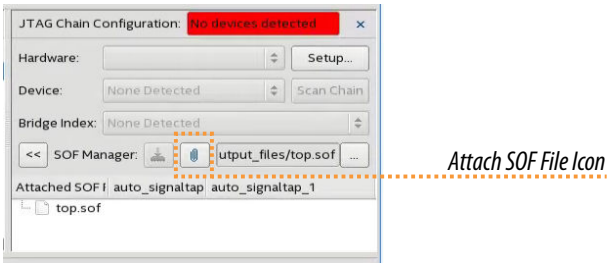


- Enable the Data Log by clicking the check box next to **Data Log**.
- The Data Log highlights the active configuration in the `.stp` file with a blue square around the active signal. The current signal list and trigger conditions appear in the **Setup** tab of the **Signal Tap** window.
- To toggle between the active configurations, double-click an entry in the Data Log.
- To store a configuration in the Data Log, click **Edit ➤ Save to Data Log**, or click the **Save to Data Log** icon at the top of the Data Log
- The time stamp in the Data Log entries displays the wall-clock time when the Signal Tap Logic Analyzer triggered, and the time elapsed from start acquisition to trigger activation.

Figure 154. SOF Manager

The SOF Manager is in the **JTAG Chain Configuration** pane.

With the SOF Manager you can embed multiple SOFs into one `.stp` file. This action lets you move the `.stp` file to a different location, either on the same computer or across a network, without including the associated `.sof` separately. To embed a new SOF in the `.stp` file, click the **Attach SOF File** icon .



As you switch between configurations in the Data Log, you can extract the SOF that is compatible with that configuration.

To download the new SOF to the FPGA, click the Program Device icon in the SOF Manager, after ensuring that the configuration of your `.stp` matches the design programmed into the target device.

8.5 Define Triggers

Specify various types of trigger conditions using the Signal Tap Logic Analyzer on the **Signal Configuration** pane. When you start the Signal Tap Logic Analyzer, it samples activity continuously from the monitored signals. The Signal Tap Logic Analyzer "triggers"—that is, the logic analyzer stops and displays the data—when a condition or set of conditions that you specified have been reached.

8.5.1 Basic Trigger Conditions

If you select the **Basic AND** or **Basic OR** trigger type, you must specify the trigger pattern for each signal you have added in the `.stp`. To specify the trigger pattern, right-click the **Trigger Conditions** column and click the desired pattern. Set the trigger pattern to any of the following conditions:

- **Don't Care**
- **Low**
- **High**
- **Falling Edge**
- **Rising Edge**
- **Either Edge**

For buses, type a pattern in binary, or right-click and select **Insert Value** to enter the pattern in other number formats. Note that you can enter X to specify a set of "don't care" values in either your hexadecimal or your binary string. For signals in the `.stp` file that have an associated mnemonic table, you can right-click and select an entry from the table to specify pre-defined conditions for the trigger.

When you add signals through plug-ins, you can create basic triggers using predefined mnemonic table entries. For example, with the Nios II plug-in, if you specify an `.elf` file from your Nios II IDE design, you can type the name of a function from your Nios II code. The logic analyzer triggers when the Nios II instruction address matches the address of the code function name that you specify.

Data capture stops and the Logic Analyzer stores the data in the buffer when the logical AND of all the signals for a given trigger condition evaluates to TRUE.

Related Links

View, Analyze, and Use Captured Data on page 313
Use the Signal Tap Logic Analyzer interface to examine the data you captured manually or using a trigger, and use your findings to debug your design.

8.5.1.1 Using the Basic OR Trigger Condition with Nested Groups

When you specify a set of signals as a nested group (group of groups) with the **Basic OR** trigger type, Signal Tap Logic Analyzer generates an advanced trigger condition. This condition sorts signals within groups to minimize the need to recompile your

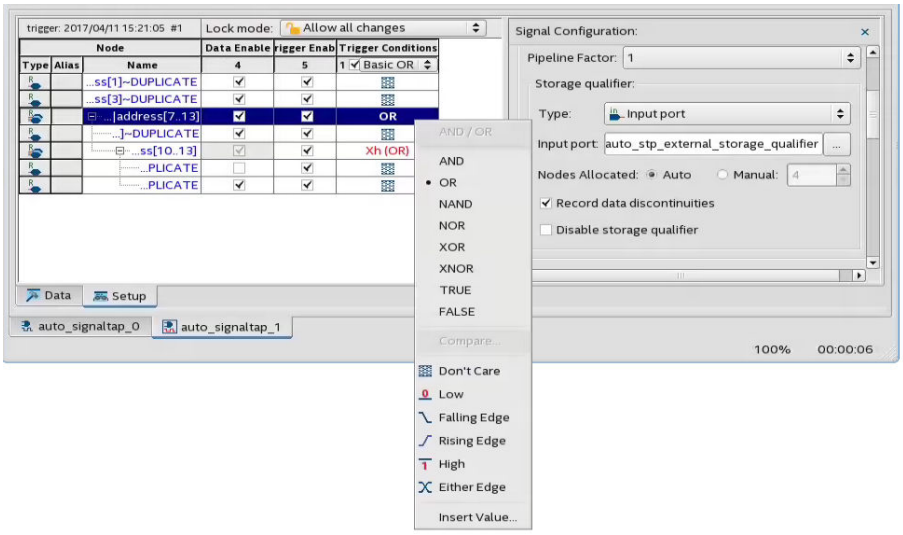
design. As long as the parent-child relationships of nodes are kept constant, the advanced trigger condition does not change. You can modify the sibling relationships of nodes and not need to recompile your design.

The evaluation precedence of a nested trigger condition starts at the bottom-level with the leaf-groups. The Logic Analyzer uses the resulting logic value to compute the parent group's logic value. If you manually set the value of a group, the logic value of the group's members doesn't influence the result of the group trigger. To create a nested trigger condition:

1. Select **Basic OR** under **Trigger Conditions**.
2. In the **Setup** tab, select several nodes. Include groups in your selection.
3. Right-click the **Setup** tab and select **Group**.
4. Select the nested group and right-click to set a group trigger condition that applies the reduction **AND**, **OR**, **NAND**, **NOR**, **XOR**, **XNOR**, or logical **TRUE** or **FALSE**.

   Note: You can only select OR and AND group trigger conditions for bottom-level groups (groups with no groups as children).

Figure 155. Applying Trigger Condition to Nested Group



8.5.2 Comparison Trigger Conditions

The **Comparison** trigger allows you to compare multiple grouped bits of a bus to an expected integer value by specifying simple comparison conditions on the bus node. The **Comparison** trigger preserves all the trigger conditions that the **Basic OR** trigger includes. You can use the **Comparison** trigger in combination with other triggers. You can also switch between **Basic OR** trigger and **Comparison** trigger at run-time, without the need for recompilation.

Signal Tap Logic Analyzer supports the following types of **Comparison** trigger conditions:

- **Single-value comparison**—compares a bus node's value to a numeric value that you specify. Use one of these operands for comparison: >, >=, ==, <=, <. Returns 1 when the bus node matches the specified numeric value.
- **Interval check**—verifies whether a bus node's value confines to an interval that you define. Returns 1 when the bus node's value lies within the specified bounded interval.

Follow these rules when using the **Comparison** trigger condition:

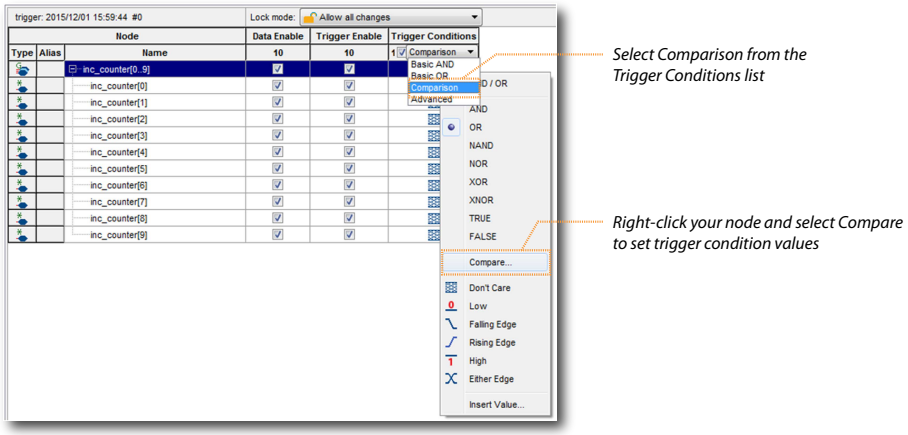
- Apply the **Comparison** trigger only to bus nodes consisting of leaf nodes.
- Do not form sub-groups within a bus node.
- Do not enable or disable individual trigger nodes within a bus node.
- Do not specify comparison values (in case of single-value comparison) or boundary values (in case of interval check) exceeding the selected node's bus-width.

8.5.2.1 Specifying the Comparison Trigger Conditions

Follow these steps to specify the **Comparison** trigger conditions:

1. From the **Setup** tab, select **Comparison** under **Trigger Conditions**.
2. Right-click the node in the trigger editor, and select **Compare**.

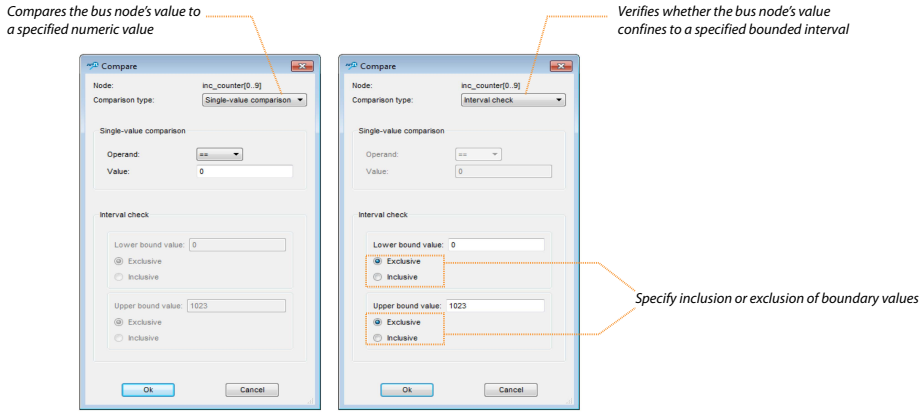
Figure 156. Selecting the Comparison Trigger Condition



3. Select the **Comparison type** from the Compare window.
   - If you choose **Single-value comparison** as your comparison type, specify the operand and value.
   - If you choose **Interval check** as your comparison type, provide the lower and upper bound values for the interval.

   You can also specify if you want to include or exclude the boundary values.

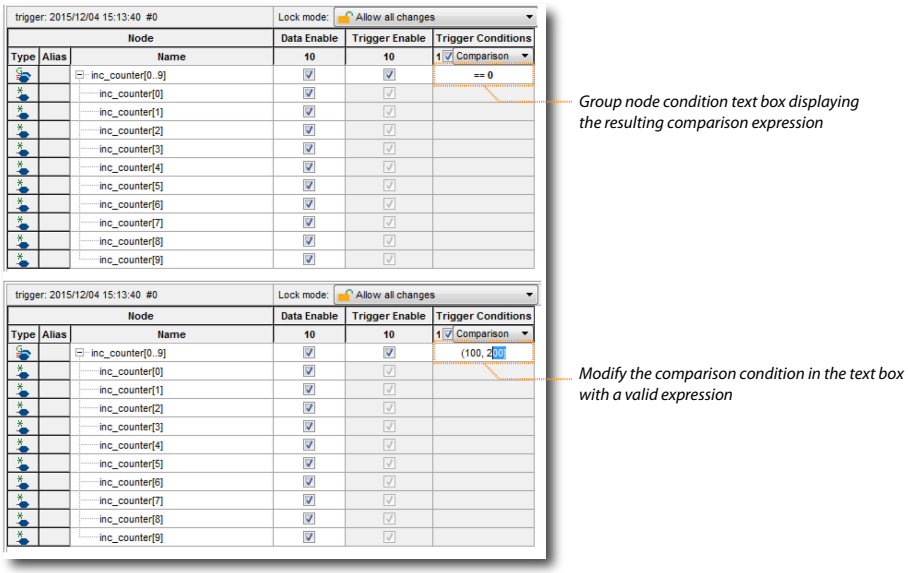
Figure 157. Specifying the Comparison Values



4. Click **OK**. The trigger editor displays the resulting comparison expression in the group node condition text box.

   Note: You can modify the comparison condition in the text box with a valid expression.

Figure 158. Resulting Comparison Condition in Text Box

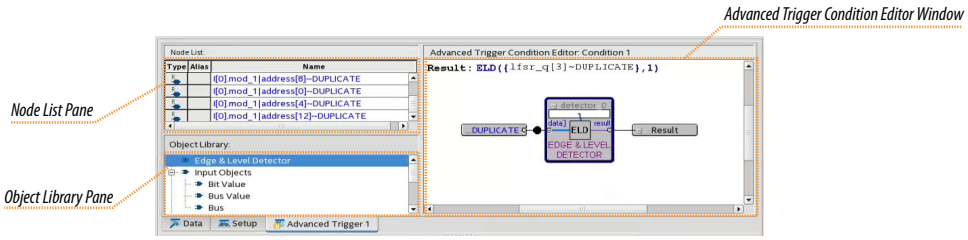


8.5.3 Advanced Trigger Conditions

To capture data for a given combination of conditions, build an advanced trigger. The Signal Tap Logic Analyzer provides the **Advanced Trigger** tab, where you can build a complex trigger expression using a GUI.

To open the **Advanced Trigger** tab, select the **Advanced** trigger type at the top of the **Trigger Conditions** column in the Node list.

Figure 159. Advanced Trigger Condition Tab



Drag-and-drop operators from the **Object Library** pane and the **Node List** pane into the **Advanced Trigger Configuration Editor** window, to build the complex trigger condition in an expression tree. To configure the operators' settings, double-click or right-click the operators that you have placed and select **Properties**.

Table 88. Advanced Triggering Operators

Name of Operator	Type
Less Than	Comparison
Less Than or Equal To	Comparison
Equality	Comparison
Inequality	Comparison
Greater Than	Comparison
Greater Than or Equal To	Comparison
Logical NOT	Logical
Logical AND	Logical
Logical OR	Logical
Logical XOR	Logical
Reduction AND	Reduction
Reduction OR	Reduction
Reduction XOR	Reduction
Left Shift	Shift
Right Shift	Shift
Bitwise Complement	Bitwise
Bitwise AND	Bitwise
Bitwise OR	Bitwise
Bitwise XOR	Bitwise
Edge and Level Detector	Signal Detection

Adding many objects to the Advanced Trigger Condition Editor can make the work space cluttered and difficult to read. To keep objects organized while you build your advanced trigger condition, use the shortcut menu and select **Arrange All Objects**. Alternatively, use the **Zoom-Out** command to fit more objects into the **Advanced Trigger Condition Editor** window.

8.5.3.1 Examples of Advanced Triggering Expressions

The following examples show how to use Advanced Triggering:

Figure 160. Bus outa Is Greater Than or Equal to Bus outb

Trigger when bus `outa` is greater than or equal to `outb`.

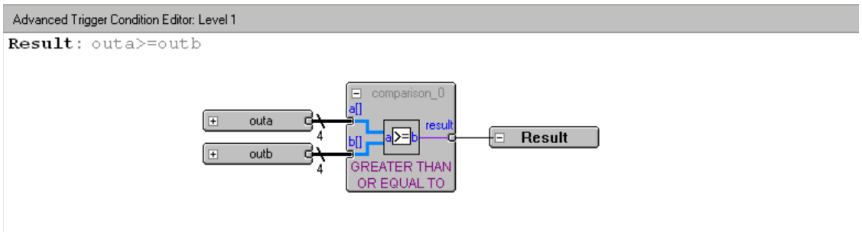


Figure 161. Enable Signal Has a Rising Edge

Trigger when bus `outa` is greater than or equal to bus `outb`, and when the enable signal has a rising edge.

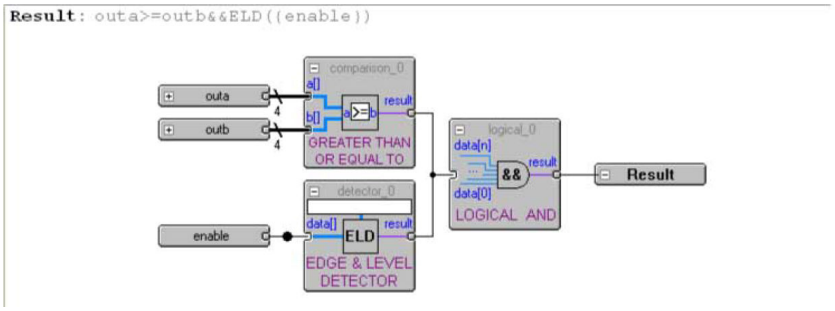
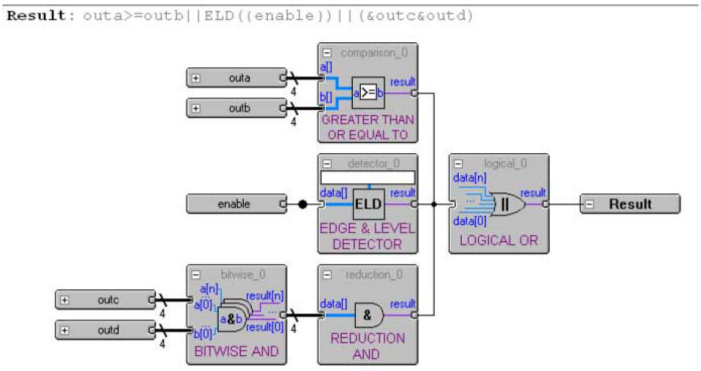


Figure 162. Bitwise AND Operation

Trigger when bus `outa` is greater than or equal to bus `outb`, or when the enable signal has a rising edge. Or, when a bitwise AND operation has been performed between bus `outc` and bus `outd`, and all bits of the result of that operation are equal to 1.

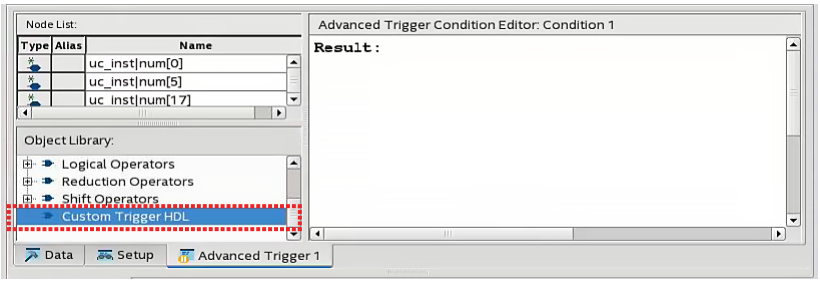


8.5.4 Custom Trigger HDL Object

The Custom Trigger HDL object allows you to create a custom trigger condition with your own HDL module in either Verilog or VHDL. The Custom Trigger HDL object appears in the **Object Library** pane of the **Advanced Trigger** editor.

Use the Custom Trigger HDL object to simulate your triggering logic and ensure that the logic itself is not faulty. You can tap specific instances of modules located anywhere in the hierarchy of your design, without having to manually route all the necessary connections.

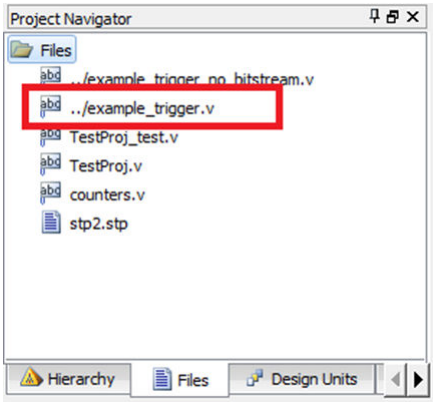
Figure 163. Object Library



8.5.4.1 Custom Trigger Flow

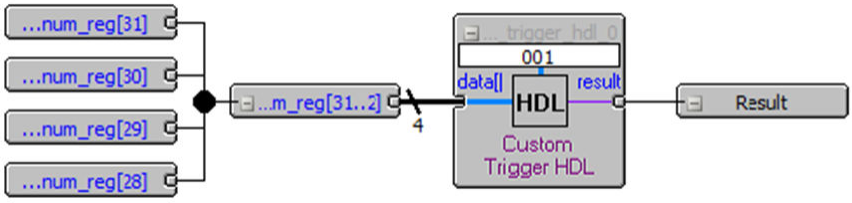
1. Select **Advanced** for a given trigger-level to make the Advanced Trigger editor active.
2. Prepare your Custom Trigger HDL module. You can either add a new source file to Quartus Prime that contains the trigger module or append the HDL for your trigger module to a source file already included in Quartus Prime **Files** under the **Project Navigator**.

Figure 164. Project Navigator



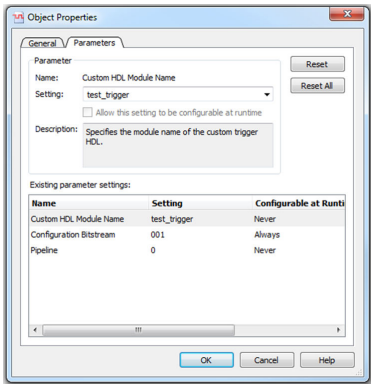
3. Implement the required inputs and outputs for your Custom Trigger HDL module.
4. Drag in your Custom Trigger HDL object and connect the object's data input bus and result output bit to the final trigger result.

Figure 165. Custom Trigger HDL Object



5. Right-click your Custom Trigger HDL object and configure the object's properties.

Figure 166. Configure Object Properties



6. Compile your design.
7. Acquire data with Signal Tap using your custom Trigger HDL object.

Table 89. Custom Trigger HDL Module Required Inputs and Outputs

Name	Description	Input/Output	Required/ Optional
`acq_clk`	Acquisition clock used by Signal Tap	Input	Required
`reset`	Reset signal used by Signal Tap when restarting a capture.	Input	Required
`data_in`	• Data input to be connected in the Advanced Trigger editor. • Data your module will use to trigger.	Input	Required
`pattern_in`	• Module's input for the configuration bitstream property. • Runtime configurable property that can be set from Signal Tap GUI to change the behavior of your trigger logic.	Input	Optional
`trigger_out`	Output signal of your module to be asserted when triggering conditions have been met.	Output	Required

Table 90. Custom Trigger HDL Module Properties

Property	Description
Custom HDL Module Name	Module name of your triggering logic

continued...

Property	Description
	i.e. module **trigger_foo** (input x, y ...);
Configuration Bitstream	• Allows you to create runtime-configurable trigger logic which can change its behavior based upon the value of the configuration bitstream. • Configuration bitstream property is interpreted as binary and should only contain the characters 1 and 0. The bit-width (number of 1s and 0s) should match the `pattern_in` bit width. • A blank configuration bitstream implies that your module does not have a `pattern_in` input.
Pipeline	• Tells the advanced trigger editor how many stages of pipeline your triggering logic has. • If it takes three clock cycles after a triggering input is received for the trigger output to be asserted, you can denote a pipeline value of three.

Figure 167. Example of Verilog Trigger Using Configuration Bitstream

```
module test_trigger(input acq_clk, reset, input [3:0] data_in, input
[1:0] pattern_in, output reg trigger_out);

    always @ (pattern_in) begin
        case (pattern_in)
            2'b00:
                trigger_out = &data_in;
            2'b01:
                trigger_out = |data_in;
            2'b10:
                trigger_out = 1'b0;
            2'b11:
                trigger_out = 1'b1;
        endcase
    end

endmodule
```

Figure 168. Example of Verilog Trigger with No Configuration Bitstream

```
module test_trigger_no_bs(input acq_clk, reset, input [3:0]
data_in, output trigger_out);

    assign trigger_out = &data_in;

endmodule
```

8.5.5 Trigger Condition Flow Control

The Signal Tap Logic Analyzer offers multiple triggering conditions to give you precise control of the method to capture data into the acquisition buffers. The Trigger Condition Flow allows you to define the relationship between a set of triggering conditions. The Signal Tap Logic Analyzer **Signal Configuration** pane offers two flow control mechanisms for organizing trigger conditions:

- **Sequential Triggering**—the default triggering flow. Sequential triggering allows you to define up to 10 triggering levels that must be satisfied before the acquisition buffer finishes capturing.
- **State-Based Triggering**—allows you the greatest control over your acquisition buffer. Custom-based triggering allows you to organize trigger conditions into states based on a conditional flow that you define.

You can use sequential or state based triggering with either a segmented or a non-segmented buffer.

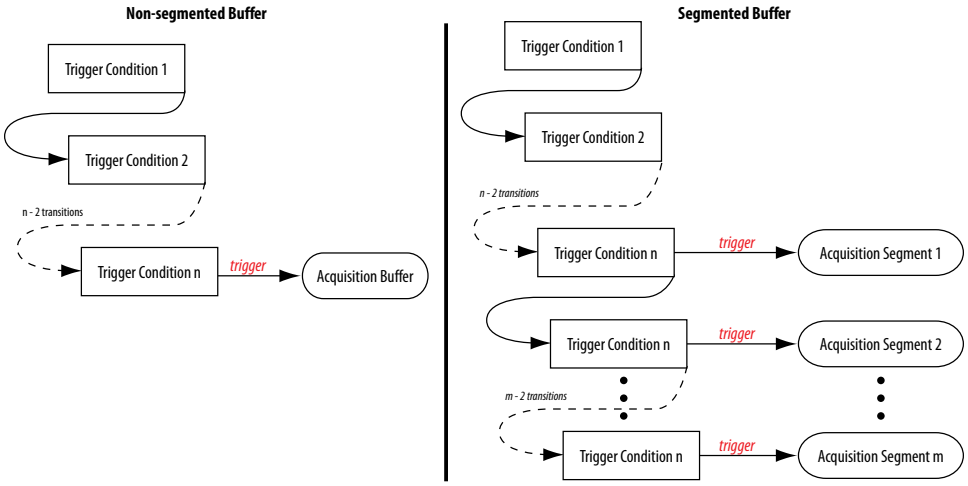
8.5.5.1 Sequential Triggering

Sequential triggering flow allows you to cascade up to 10 levels of triggering conditions. Signal Tap Logic Analyzer sequentially evaluates each of the conditions.

When the last triggering condition evaluates to TRUE, the Signal Tap Logic Analyzer triggers the acquisition buffer. For segmented buffers, every acquisition segment after the first triggers on the last condition that you specified. Use the Simple Sequential Triggering feature with basic triggers, comparison triggers, advanced triggers, or a mix of all three. The figure illustrates the simple sequential triggering flow for non-segmented and segmented buffers.

The external trigger is considered as trigger level 0. The external trigger must be evaluated before the main trigger levels are evaluated.

Figure 169. Sequential Triggering Flow



Notes to figure:

1. The acquisition buffer starts capture when all n triggering levels are satisfied, where $n \leq 10$.
2. If you define an external trigger input, the Logic Analyzer evaluates it before evaluating all other trigger conditions.

To configure the Signal Tap Logic Analyzer for sequential triggering, in the Signal Tap editor on the **Trigger flow control** list, select **Sequential**. Select the desired number of trigger conditions from the **Trigger Conditions** list. After you select the trigger conditions, configure each trigger condition in the node list. You can enable/disable any trigger condition from the top of the column in the node list.

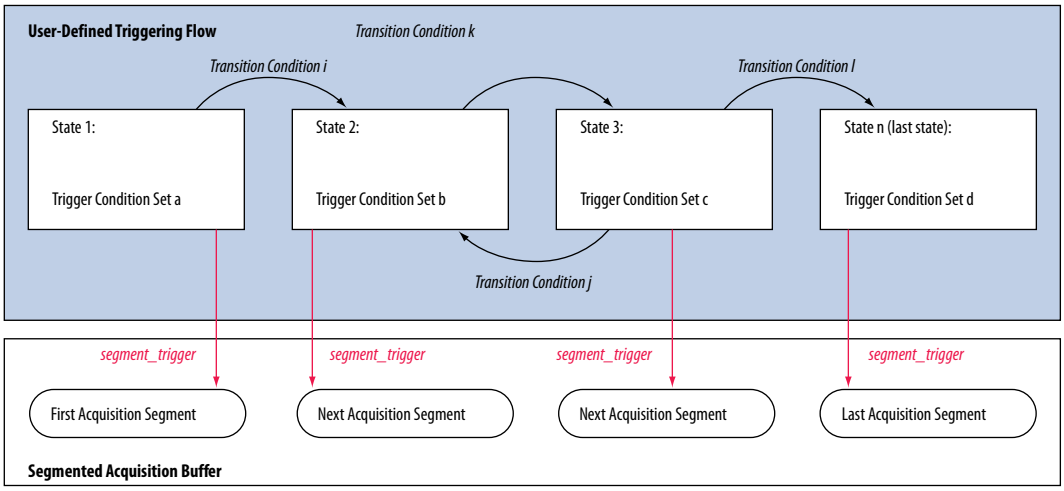
8.5.5.2 State-Based Triggering

Custom State-based triggering provides the most control over triggering condition arrangement. The State-Based Triggering flow allows you to describe the relationship between triggering conditions precisely, using the GUI and the Signal Tap Trigger Flow Description Language, a description language based upon conditional expressions.

Tooltips within the custom triggering flow GUI help you to describe your desired flow. The custom State-based triggering flow allows for more efficient use of the space available in the acquisition buffer, because the Logic Analyzer captures only specific samples of interest.

With state-based triggering, you define a state diagram to organize the events that trigger the acquisition buffer. The states capture all actions that the acquisition buffer performs, and the conditional expressions that you specify within each state define all transition conditions between the states.

Figure 170. State-Based Triggering Flow



Notes to figure:

1. You can define up to 20 different states.
2. An external trigger input, if defined, is evaluated before any conditions in the custom State-based triggering flow are evaluated.

Each state allows you to define a set of conditional expressions. Each conditional expression is a Boolean expression dependent on a combination of triggering conditions (that you configure within the **Setup** tab), counters, and status flags. The Signal Tap Logic Analyzer custom-based triggering flow provides counters and status flags.

Within each conditional expression you define a set of actions. Actions include triggering the acquisition buffer to stop capture, a modification to either a counter or status flag, or a state transition.

Trigger actions can apply to either a single segment of a segmented acquisition buffer or to the entire non-segmented acquisition buffer. Each trigger action provides you with an optional count that specifies the number of samples the buffer captures before

the logic analyzer stops acquisition of the current segment. The count argument allows you to control the amount of data the buffer captures before and after triggering event.

Resource manipulation actions allow you to increment and decrement counters or set and clear status flags. The logic analyzer uses counter and status flag resources as optional inputs in conditional expressions. Counters and status flags are useful for counting the number of occurrences of certain events and for aiding in triggering flow control.

The Signal Tap custom State-Based triggering flow allows you to capture a sequence of events that may not necessarily be contiguous in time. For example, a communication transaction between two devices that includes a hand shaking protocol containing a sequence of acknowledgements.

The **State-Based Trigger Flow** tab is the control interface for the custom state-based triggering flow. To enable this tab, select **State-based** on the **Trigger Flow Control** list. When you specify **Trigger Flow Control** as **Sequential**, the **State-Based Trigger Flow** tab is hidden.

The **State-Based Trigger Flow** tab contains three panes:

- **State Diagram** pane
- **Resources** pane
- **State Machine** pane

8.5.5.2.1 State Diagram Pane

The **State Diagram** pane provides a graphical overview of the triggering flow that you define. It shows the number of states available and the state transitions between the states. You can adjust the number of available states by using the menu above the graphical overview.

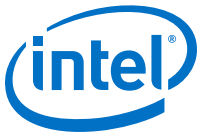
8.5.5.2.2 State Machine Pane

The **State Machine** pane contains the text entry boxes where you can define the triggering flow and actions associated with each state. You can define the triggering flow using the Signal Tap Trigger Flow Description Language, a simple language based on “if-else” conditional statements. Tooltips appear when you move the mouse over the cursor, to guide command entry into the state boxes. The GUI provides a syntax check on your flow description in real-time and highlights any errors in the text flow.

The State Machine description text boxes default to show one text box per state. You can also have the entire flow description shown in a single text field. This option can be useful when copying and pasting a flow description from a template or an external text editor. To toggle between one window per state, or all states in one window, select the appropriate option under **State Display mode.**

Related Links

Signal Tap Trigger Flow Description Language on page 296
The Trigger Flow Description Language is based on a list of conditional expressions per state to define a set of actions.



8.5.5.2.3 Resources Pane

The **Resources** pane allows you to declare Status Flags and Counters for use in the conditional expressions in the Custom Triggering Flow. Actions to decrement and increment counters or to set and clear status flags are performed within the triggering flow that you define.

You can specify up to 20 counters and 20 status flags. Counter and status flags values may be initialized by right-clicking the status flag or counter name after selecting a number of them from the respective pull-down list, and selecting **Set Initial Value.** To specify a counter width, right-click the counter name and select **Set Width**. Counters and flag values are updated dynamically after acquisition has started to assist in debugging your trigger flow specification.

The **configurable at runtime** options in the **Resources** pane allows you to configure the custom-flow control options that can be changed at runtime without requiring a recompilation.

Table 91. Runtime Reconfigurable Settings, State-Based Triggering Flow

Setting	Description
Destination of goto action	Allows you to modify the destination of the state transition at runtime.
Comparison values	Allows you to modify comparison values in Boolean expressions at runtime. In addition, you can modify the `segment_trigger` and trigger action post-fill count argument at runtime.
Comparison operators	Allows you to modify the operators in Boolean expressions at runtime.
Logical operators	Allows you to modify the logical operators in Boolean expressions at runtime.

You can restrict changes to your Signal Tap configuration to include only the options that do not require a recompilation. Trigger lock-mode allows you to make changes that can be immediately reflected in the device.

1. On the **Setup** tab, select **Allow trigger condition changes only**.
2. Modify the Trigger Flow conditions in the **Custom Trigger Flow** tab.
3. Click the desired parameter in the text box and select a new parameter from the menu that appears.

Note: Trigger lock mode restricts changes to the configuration settings that have **configurable at runtime** specified. The runtime configurable settings for the **Custom Trigger Flow** tab are on by default. You may get some performance advantages by disabling some of the runtime configurable options.

Related Links

- Performance and Resource Considerations on page 308
  There is a necessary trade-off between the runtime flexibility of the Signal Tap Logic Analyzer, the timing performance of the Signal Tap Logic Analyzer, and resource usage.
- Runtime Reconfigurable Options on page 311

8.5.5.3 Signal Tap Trigger Flow Description Language

The Trigger Flow Description Language is based on a list of conditional expressions per state to define a set of actions. Each line in the example shows a language format. Keywords are shown in bold. Non-terminals are delimited by "<>" and are further explained in the following sections. Optional arguments are delimited by "[]".

```
state <State_label>:
<action_list>

if( <Boolean_expression> )
<action_list>
[else if ( <boolean_expression> )
<action_list>]
[else
<action_list>]
```

Note: Multiple `else if` conditions are allowed.

The priority for evaluation of conditional statements is assigned from top to bottom. The *<boolean_expression>* in an `if` statement can contain a single event, or it can contain multiple event conditions. The `action_list` within an `if` or an `else if` clause must be delimited by the begin and end tokens when the action list contains multiple statements. When the boolean expression is evaluated TRUE, the logic analyzer analyzes all of the commands in the action list concurrently. The possible actions include:

- Triggering the acquisition buffer
- Manipulating a counter or status flag resource
- Defining a state transition

Related Links

Custom Triggering Flow Application Examples on page 331
The custom triggering flow in the Signal Tap Logic Analyzer is most useful for organizing a number of triggering conditions and for precise control over the acquisition buffer.

8.5.5.4 State Labels

State labels are identifiers that can be used in the action `goto`.

`state` *<state_label>*: begins the description of the actions evaluated when this state is reached.

The description of a state ends with the beginning of another state or the end of the whole trigger flow description.

8.5.5.5 Boolean_expression

`Boolean_expression` is a collection of logical operators, relational operators, and their operands that evaluate into a Boolean result. Depending on the operator, the operand can be a reference to a trigger condition, a counter and a register, or a numeric value. Within an expression, parentheses can be used to group a set of operands.

Logical operators accept any boolean expression as an operand.

Table 92. Logical Operators

Operator	Description	Syntax
!	NOT operator	`! expr1`
&&	AND operator	`expr1 && expr2`
||	OR operator	`expr1 || expr2`

Relational operators are performed on counters or status flags. The comparison value, the right operator, must be a numerical value.

Table 93. Relational Operators

Operator	Description	Syntax
>	Greater than	*<identifier> > <numerical_value>*
>=	Greater than or Equal to	*<identifier> >= <numerical_value>*
==	Equals	*<identifier> == <numerical_value>*
!=	Does not equal	*<identifier> != <numerical_value>*
<=	Less than or equal to	*<identifier> <= <numerical_value>*
<	Less than	*<identifier> < <numerical_value>*

Notes to table:
1. *<identifier>* indicates a counter or status flag.
2. *<numerical_value>* indicates an integer.

8.5.5.6 Action_list

Action_list is a list of actions that can be performed when a state is reached and a condition is also satisfied. If more than one action is specified, they must be enclosed by `begin` and `end`. The actions can be categorized as resource manipulation actions, buffer control actions, and state transition actions. Each action is terminated by a semicolon (`;`).

8.5.5.7 Resource Manipulation Action

The resources used in the trigger flow description can be either counters or status flags.

Table 94. Resource Manipulation Action

Action	Description	Syntax
increment	Increments a counter resource by 1	`increment <counter_identifier>;`
decrement	Decrements a counter resource by 1	`decrement <counter_identifier>;`
reset	Resets counter resource to initial value	`reset <counter_identifier>;`
set	Sets a status Flag to 1	`set <register_flag_identifier>;`
clear	Sets a status Flag to 0	`clear <register_flag_identifier>;`

8.5.5.8 Buffer Control Action

Buffer control actions specify an action to control the acquisition buffer.

Table 95. Buffer Control Action

Action	Description	Syntax
`trigger`	Stops the acquisition for the current buffer and ends analysis. This command is required in every flow definition.	`trigger <post-fill_count>;`
`segment_trigger`	Ends the acquisition of the current segment. The Signal Tap Logic Analyzer starts acquiring from the next segment on evaluating this command. If all segments are filled, the oldest segment is overwritten with the latest sample. The acquisition stops when a trigger action is evaluated. This action cannot be used in non-segmented acquisition mode.	`segment_trigger <post-fill_count>;`
`start_store`	Asserts the `write_enable` to the Signal Tap acquisition buffer. This command is active only when the State-based storage qualifier mode is enabled.	`start_store`
`stop_store`	De-asserts the `write_enable` signal to the Signal Tap acquisition buffer. This command is active only when the State-based storage qualifier mode is enabled.	`stop_store`

Both `trigger` and `segment_trigger` actions accept an optional post-fill count argument. If provided, the current acquisition acquires the number of samples provided by post-fill count and then stops acquisition. If no post-count value is specified, the trigger position for the affected buffer defaults to the trigger position specified in the **Setup** tab.

Note: In the case of `segment_trigger`, acquisition of the current buffer stops immediately if a subsequent triggering action is issued in the next state, regardless of whether or not the post-fill count has been satisfied for the current buffer. The remaining unfilled post-count acquisitions in the current buffer are discarded and displayed as grayed-out samples in the data window.

8.5.5.9 State Transition Action

The State Transition action specifies the next state in the custom state control flow. It is specified by the `goto` command. The syntax is as follows:

```
goto <state_label>;
```

8.5.5.10 Using the State-Based Storage Qualifier Feature

When you select State-based for the storage qualifier type, the State-based trigger flow enables the `start_store` and `stop_store` actions. When you use these commands in conjunction with the expressions of the State-based trigger flow, you get maximum flexibility to control data written into the acquisition buffer.

Note: You can only apply the `start_store` and `stop_store` commands to a non-segmented buffer.

The `start_store` and `stop_store` commands are similar to the start and stop conditions of the **start/stop** storage qualifier mode. If you enable storage qualification, in that Signal Tap Logic Analyzer doesn't write data into the acquisition buffer until the `start_store` command occurs. However, in the state-based storage qualifier type you must include a `trigger` command as part of the trigger flow description. This `trigger` command is necessary to complete the acquisition and display the results on the waveform display.

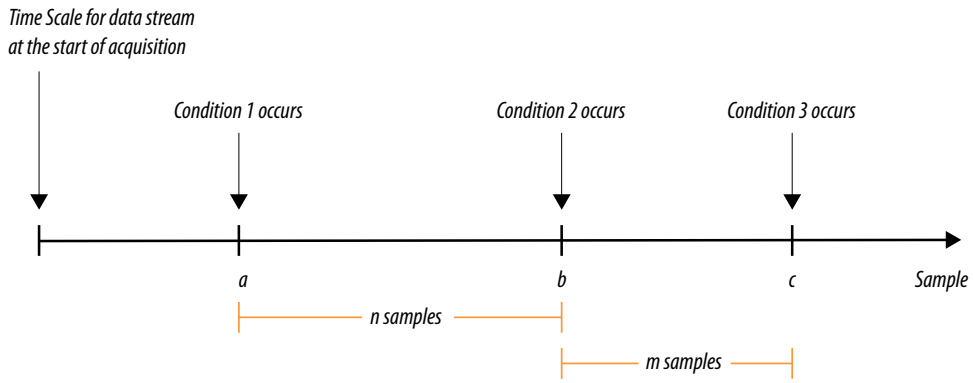
Example 28. storage qualification feature for the state-based trigger flow.

The example illustrates the behavior of the State-based trigger flow with the storage qualification commands.

This trigger flow description contains three trigger conditions that happen at different times after you click **Start Analysis**.

```
State 1: ST1:
if ( condition1 )
   start_store;
else if ( condition2 )
   trigger value;
else if ( condition3 )
   stop_store;
```

Figure 171. Capture Scenario for Storage Qualification with the State-Based Trigger Flow



When you apply the trigger flow to the scenario in the figure:

- The Signal Tap Logic Analyzer does not write into the acquisition buffer until sample **a**, when **Condition 1** occurs.
- Once sample **b** is reached, the logic analyzer evaluates the `trigger value` command, and continues to write into the buffer to finish the acquisition.
- The trigger flow specifies a `stop_store` command at sample **c**, which occurs *m* samples after the trigger point.
- If it can successfully finish the post-fill acquisition samples before **Condition 3** occurs, the logic analyzer finishes the acquisition and displays the contents of the waveform In this case, the capture ends if the post-fill count value is less than *m*.
- If the post-fill count value in the Trigger Flow description 1 is greater than *m* samples, the buffer pauses acquisition indefinitely, provided there is no recurrence of Condition 1 to trigger the logic analyzer to start capturing data again.

The Signal Tap Logic Analyzer continues to evaluate the *stop_store* and *start_store* commands even after it evaluates the **trigger**. If the acquisition has paused, click **Stop Analysis** to manually stop and force the acquisition to trigger. You can use counter values, flags, and the State diagram to help you perform the trigger flow. The counter values, flags, and the current state update in real-time during a data acquisition.

The following figures show real data acquisition of the previous scenario.

Figure 172. Storage Qualification with Post-Fill Count Value Less than *m* (Acquisition Successfully Completes)

In the figure the data capture finishes successfully. It uses a buffer with a sample depth of 64, *m* = *n* = 10, and the post-fill count value = 5.

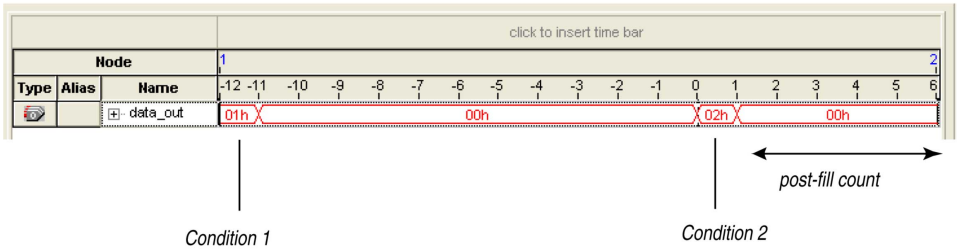


Figure 173. Storage Qualification with Post-Fill Count Value Greater than *m* (Acquisition Indefinitely Paused)

In this scenario the logic analyzer pauses indefinitely, even after a trigger condition occurs due to a `stop_store` condition. This scenario uses a sample depth of 64, with $m = n = 10$ and post-fill count = 15.

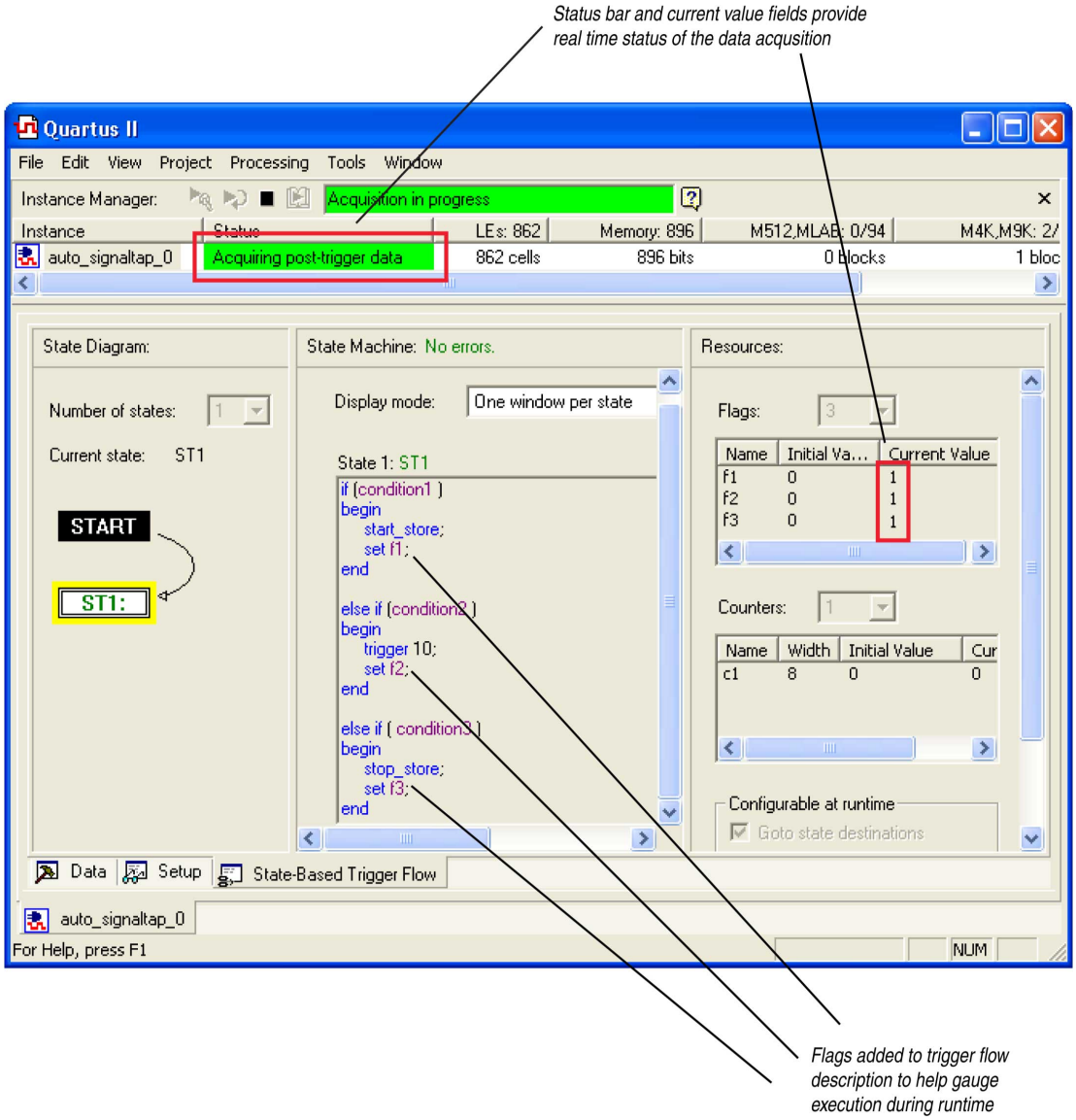
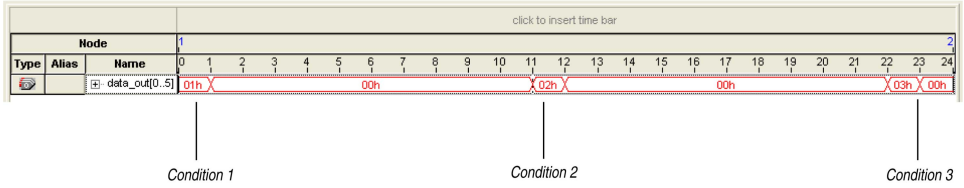


Figure 174. Waveform After Forcing the Analysis to Stop



The combination of using counters, Boolean and relational operators in conjunction with the `start_store` and `stop_store` commands can give a clock-cycle level of resolution to controlling the samples that are written into the acquisition buffer.

This code example shows a trigger flow description that skips three clock cycles of samples after hitting condition 1

```
State 1: ST1
start_store
if ( condition1 )
begin
   stop_store;
   goto ST2;
end

State 2: ST2
if (c1 < 3)
   increment c1; //skip three clock cycles; c1 initialized to 0
else if (c1 == 3)
begin
   start_store; //start_store necessary to enable writing to finish
             //acquisition
   trigger;
end
```

The following figures show the data transaction on a continuous capture and the data capture with the Trigger flow description applied.

Figure 175. Continuous Capture of Data Transaction for Example 2

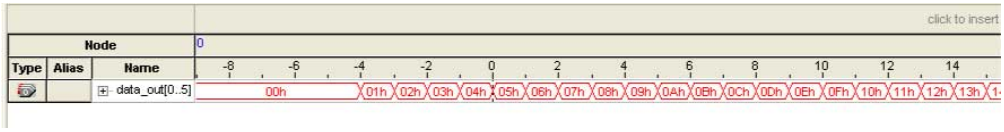
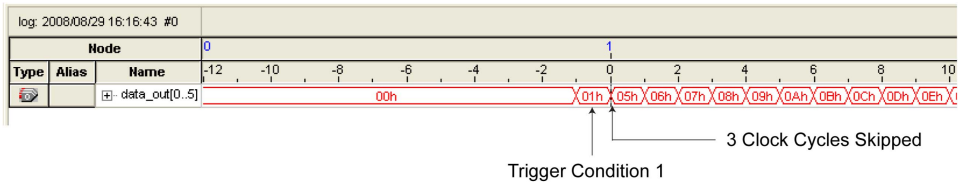


Figure 176. Capture of Data Transaction with Trigger Flow Description Applied



8.5.6 Specify Trigger Position

Specify the amount of data acquired before and after a trigger event. You can specify the trigger position independently between a Runtime and Power-Up Trigger. Select the desired ratio of pre-trigger data to post-trigger data by choosing one of the following ratios:

- **Pre**—Saves signal activity that occurred after the trigger (12% pre-trigger, 88% post-trigger).
- **Center**—Saves 50% pre-trigger and 50% post-trigger data.
- **Post**—Saves signal activity that occurred before the trigger (88% pre-trigger, 12% post-trigger).

These pre-defined ratios apply to both non-segmented buffers and segmented buffers.

If you use the custom-state based triggering flow, you can specify a custom trigger position. The `segment_trigger` and trigger actions accept a post-fill count argument. The post-fill count specifies the number of samples to capture before stopping data acquisition for the non-segmented buffer or a data segment when using the `trigger` and `segment_trigger` commands, respectively. When the captured data is displayed in the Signal Tap data window, the trigger position appears as the number of post-count samples from the end of the acquisition segment or buffer.

Sample Number of Trigger Position = (*N* – Post-Fill Count)

In this case, *N* is the sample depth of either the acquisition segment or non-segmented buffer.

For segmented buffers, the acquisition segments that have a post-count argument define use of the post-count setting. Segments that do not have a post-count setting default to the trigger position ratios defined in the **Setup** tab.

Related Links

State-Based Triggering on page 293
Custom State-based triggering provides the most control over triggering condition arrangement.

8.5.7 Create a Power-Up Trigger

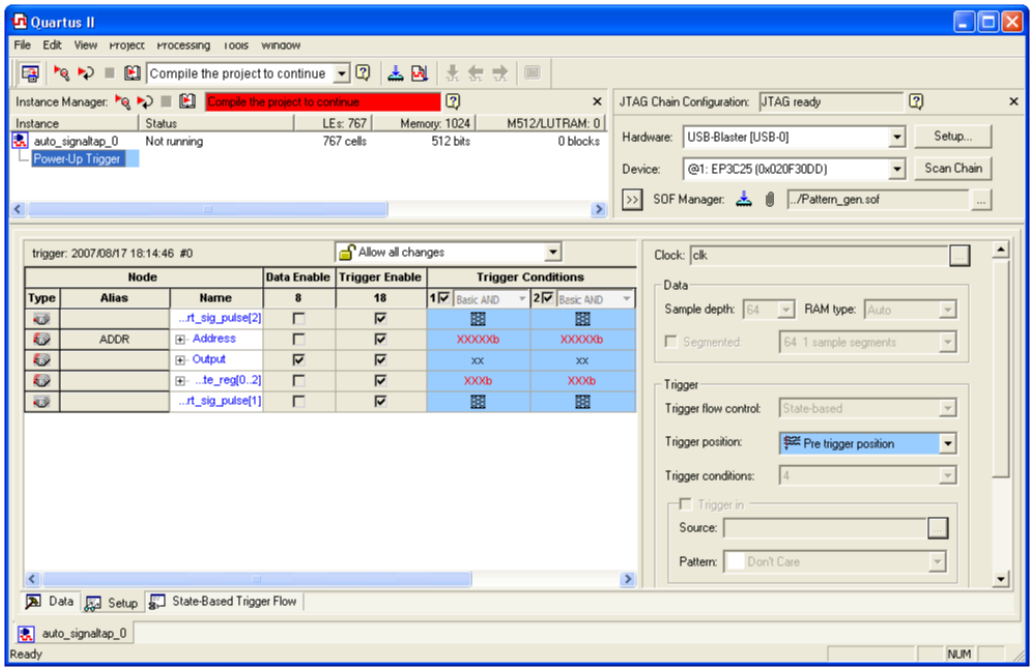
Capture trigger events that occur during device initialization, immediately after the FPGA is powered on or reset.

Typically, the Signal Tap Logic Analyzer is used to trigger on events that occur during normal device operation. You start an analysis manually once the target device is fully powered on and the JTAG connection for the device is available. With the Signal Tap Power-Up Trigger feature, you arm the Signal Tap Logic Analyzer and capture data immediately after device programming.

8.5.7.1 Enabling a Power-Up Trigger

You can add a different Power-Up Trigger to each logic analyzer instance in the **Signal Tap Instance Manager** pane. To enable the Power-Up Trigger for a logic analyzer instance, right-click the instance and click **Enable Power-Up Trigger**, or select the instance, and on the **Edit** menu, click **Enable Power-Up Trigger**. To disable a Power-Up Trigger, click **Disable Power-Up Trigger** in the same locations. Power-Up Trigger is shown as a child instance below the name of the selected instance with the default trigger conditions specified in the node list.

Figure 177. Signal Tap Logic Analyzer Editor with Power-Up Trigger Enabled



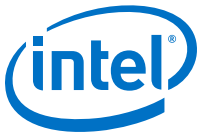
8.5.7.2 Manage and Configure Power-Up and Runtime Trigger Conditions

When the Power-Up Trigger is enabled for a logic analyzer instance, you can create basic, comparison, and advanced trigger conditions for the trigger as you do with a Run-Time Trigger. Power-Up Trigger conditions that you can adjust are color coded light blue, while Run-Time Trigger conditions you cannot adjust remain white. Since each instance now has two sets of trigger conditions—the Power-Up Trigger and the Run-Time Trigger—you can differentiate between the two with color coding. To switch between the trigger conditions of the Power-Up Trigger and the Run-Time Trigger, double-click the instance name or the Power-Up Trigger name in the **Instance Manager**.

You cannot make changes to Power-Up Trigger conditions that would normally require a full recompile with Runtime Trigger conditions, such as adding signals, deleting signals, or changing between basic, comparison, and advanced triggers. To apply these changes to the Power-Up Trigger conditions, first make the changes using the Runtime Trigger conditions.

Note: Any change made to the Power-Up Trigger conditions requires that you recompile the Signal Tap Logic Analyzer instance, even if a similar change to the Runtime Trigger conditions does not require a recompilation.

While creating or making changes to the trigger conditions for the Run-Time Trigger or the Power-Up Trigger, you may want to copy these conditions to the other trigger. This enables you to look for the same trigger during both power-up and runtime. To do this, right-click the instance name or the Power-Up Trigger name in the **Instance Manager** and click **Duplicate Trigger**, or select the instance name or the Power-Up Trigger name and on the **Edit** menu, click **Duplicate Trigger**.



You can also use In-System Sources and Probes in conjunction with the Signal Tap Logic Analyzer to force trigger conditions. The In-System Sources and Probes feature allows you to drive and sample values on to selected nets over the JTAG chain.

Related Links

Design Debugging Using In-System Sources and Probes on page 363
The Signal Tap Logic Analyzer and Signal Probe allow you to read or "tap" internal logic signals during run time as a way to debug your logic design.

8.5.8 External Triggers

Create a trigger input that allows to trigger the Signal Tap Logic Analyzer from an external source. The external trigger input behaves like trigger condition 1, is evaluated, and must be TRUE before any other configured trigger conditions are evaluated. The logic analyzer supplies a signal to trigger external devices or other Signal Tap Logic Analyzer instances. These features allow you to synchronize external logic analysis equipment with the internal logic analyzer. Power-Up Triggers can use the external triggers feature, but they must use the same source or target signal as their associated Run-Time Trigger.

You can use external triggers to perform cross-triggering on a hard processor system (HPS). Use your processor debugger to configure the HPS to obey or disregard cross-trigger request from the FPGA, and to issue or not issue cross-trigger requests to the FPGA. Use your processor debugger in combination with the Signal Tap external trigger feature to develop a dynamic combination of cross-trigger behaviors. You can use the cross-triggering feature with the ARM Development Studio 5 (DS-5) software to implement a system-level debugging solution for your Intel FPGA SoC.

Related Links

- FPGA-Adaptive Software Debug and Performance Analysis white paper
- Signal Configuration Pane
  In Quartus Prime Help

8.6 Compile the Design

To incorporate the Signal Tap logic in your design and enable the JTAG connection, you must compile your project. When you add a `.stp` file to your project, the Signal Tap Logic Analyzer becomes part of your design. When you debug your design with a traditional external logic analyzer, you must often make changes to the signals you want to monitor as well as the trigger conditions.

8.6.1 Prevent Changes Requiring Recompilation

Configure the `.stp` to prevent changes that normally require recompilation. To do this, select a lock mode from above the node list in the **Setup** tab. To lock your configuration, choose to allow only trigger condition changes.

Related Links

Setup Tab (Signal Tap Logic Analyzer)
In Quartus Prime Help

8.6.2 Incremental Route with Rapid Recompile

You can use Incremental Route with Rapid Recompile to decrease compilation times. After performing a full compilation on your design, you can use the Incremental Route flow to achieve a 2-4x speedup over a flat compile. The Incremental Route flow is not compatible with Partial Reconfiguration.

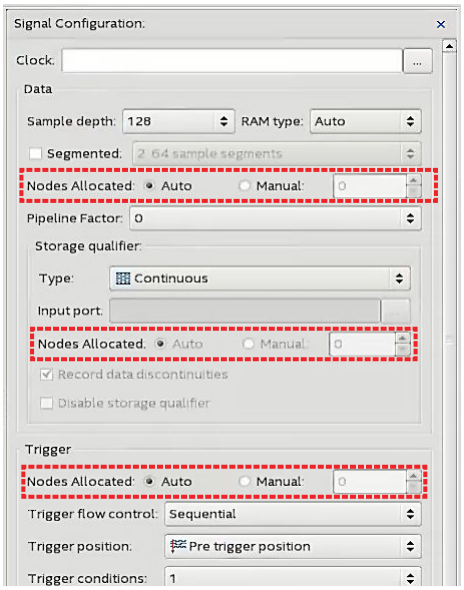
Quartus Prime Pro Edition software supports Incremental Route with Rapid Recompile for Arria 10 devices.

Related Links

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

8.6.2.1 Incremental Route Flow

Figure 178. Manually Allocate Nodes

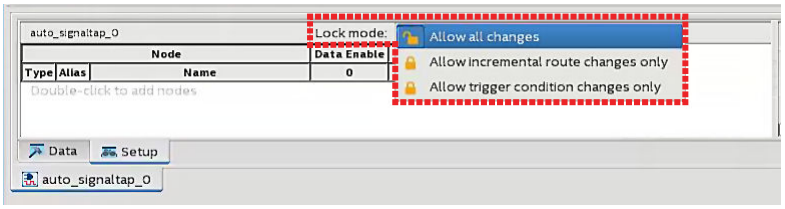


1. Open your design and run **Analysis & Elaboration** (or a full compilation) to give node visibility in Signal Tap.
2. Add Signal Tap to your design, and specify manual allocation for Trigger and Data (Storage Qualifier, if used) nodes in the Signal Tap **Signal Configuration** pane.

- Selecting **Manual** allows you to control the number of nodes compiled into the design. This is critical for the Incremental Route flow.
- If you select **Auto**, the number of nodes compiled into the design will directly reflect the number of nodes (not including groups, which are not signals) currently in the **Setup** tab. If you then add a node, the number of nodes required on the device will not match what has already been compiled, and you will then need to perform a full compilation.

3. Specify the number of nodes that you estimate will be needed for the debugging process. You can increase the number of nodes later, but this will require more compilation time.
4. Connect nodes you are interested in tapping
5. Run a full compilation, if you have not already done a full compile on your project. Otherwise, you can start incremental compile using Rapid Recompile.
6. Debug and determine additional signals of interest.
7. (Optional) Turn on **Allow incremental route changes only** lock-mode.

Figure 179. Incremental Route Lock-Mode



8. Add additional nodes in the Signal Tap **Setup** tab without exceeding the number of manually allocated nodes in step 2. Avoid making changes to non-runtime configurable settings.
9. Click the Rapid Recompile icon from the toolbar. Alternatively, click **Processing ➤ Start Rapid Recompile**.

   Note: The previous steps set up your design for Incremental Route, but the actual Incremental Route process begins when you perform a Rapid Recompile.

8.6.2.2 Tips to Achieve Maximum Speedup

- Basic AND (which applies to Storage Qualifier as well as trigger input) is the fastest for the Incremental Route flow.
- Basic OR is slower for the Incremental Route flow, but if you avoid changing the parent-child relationship of nodes within groups, you can minimize the impact on compile time. You can change the sibling relationships of nodes.
  - Basic OR and advanced triggers require re-synthesis when the number/names of tapped nodes are changed.
- Use the Incremental Route lock-mode to avoid inadvertent changes requiring a full compilation.

8.6.3 Timing Preservation with the Signal Tap Logic Analyzer

In addition to verifying functionality, timing closure is one of the most crucial processes in successful operation of your design. The Quartus Prime Pro Edition software supports timing preservation for post-fit taps in Arria 10 designs with the Rapid Recompile feature. Rapid Recompile automatically reuses verified portions of your design during recompilations, rather than reprocessing those portions.

Use the following techniques to help maintain timing:

- Avoid adding critical path signals to your `.stp`.
- Minimize the number of combinational signals you add to your `.stp`, and add registers whenever possible.
- Specify an f_{MAX} constraint for each clock in your design.

Related Links

Timing Closure and Optimization
In *Quartus Prime Pro Edition Handbook Volume 2*

8.6.4 Performance and Resource Considerations

There is a necessary trade-off between the runtime flexibility of the Signal Tap Logic Analyzer, the timing performance of the Signal Tap Logic Analyzer, and resource usage. The Signal Tap Logic Analyzer allows you to select the runtime configurable parameters to balance the need for runtime flexibility, speed, and area.

The default values of the runtime configurable parameters provide maximum flexibility, so you can complete debugging as quickly as possible; however, you can adjust these settings to determine whether there is a more optimal configuration for your design.

If Signal Tap logic is part of your critical path, follow these tips to speed up the performance of the Signal Tap Logic Analyzer:

- **Disable runtime configurable options**—Certain resources are allocated to accommodate for runtime flexibility. If you use either advanced triggers or State-based triggering flow, disable runtime configurable parameters for a boost in f_{MAX} of the Signal Tap logic.
  - If you are using State-based triggering flow, try disabling the **Goto state destination** option and performing a recompilation before disabling the other runtime configurable options. The **Goto state destination** option has the greatest impact on f_{MAX}, as compared to the other runtime configurable options.
- **Minimize the number of signals that have Trigger Enable selected**—All signals that you add to the `.stp` have **Trigger Enable** turned on. Turn off **Trigger Enable** for signals that you do not plan to use as triggers.
- **Turn on Physical Synthesis for register retiming**—If many (more than the number of inputs that fit in a LAB) enabled triggering signals fan-in logic to a gate-based triggering condition (basic trigger condition or a logical reduction operator in the advanced trigger tab), turn on **Perform register retiming**. This can help balance combinational logic across LABs.

If your design is resource constrained, follow these tips to reduce the logic or memory used by the Signal Tap Logic Analyzer:

- **Disable runtime configurable options**—Disabling runtime configurability for advanced trigger conditions or runtime configurable options in the State-based triggering flow results in fewer LEs.
- **Minimize the number of segments in the acquisition buffer**—You can reduce the logic resources that the Signal Tap Logic Analyzer uses if you limit the segments in your sampling buffer.
- **Disable the Data Enable for signals that you use only for triggering**—By default, both the **data enable** and **trigger enable** options are selected for all signals. Turning off the **data enable** option for signals you use only as trigger inputs saves on memory resources used by the Signal Tap Logic Analyzer.

Because performance results are design-dependent, try these options in different combinations until you achieve the desired balance between functionality, performance, and utilization.

8.7 Program the Target Device or Devices

After you add the Signal Tap Logic Analyzer to your project and re-compile, you can configure the FPGA target device. If you are using the Signal Tap Logic Analyzer for debugging, configure the device from the `.stp` instead of the Quartus Prime Programmer. This allows you to open more than one `.stp` file and program multiple devices, thus debugging multiple designs simultaneously.

The settings in an `.stp` must be compatible with the `.sof` file you use to program the device. A `.stp` file is compatible with a `.sof` file when the settings for the logic analyzer, such as the size of the capture buffer and the signals you use for monitoring or triggering, match the programming settings of the target device. If the files are not compatible, you can still program the device, but you cannot run or control the logic analyzer from the Signal Tap Logic Analyzer Editor.

Note: When the Signal Tap Logic Analyzer detects incompatibility after analysis is started, the Quartus Prime software generates a system error message containing two CRC values: the expected value and the value retrieved from the `.stp` instance on the device. The CRC value cimes from all Signal Tap settings that affect the compilation.

To ensure programming compatibility, program your device with the `.sof` file generated in the most recent compilation. To check whether a particular `.sof` is compatible with the current Signal Tap configuration, attach the `.sof` to the SOF manager.

Before starting a debugging session, do not make any changes to the `.stp` settings that require recompiling the project. To verify whether a change you made requires recompiling the project, check the Signal Tap status display at the top of the **Instance Manager** pane. This feature allows you to undo the change, so that you do not need to recompile your project. To prevent these types of changes, select **Allow trigger condition changes only** to lock the `.stp`. The Incremental Route lock mode, **Allow incremental route changes only**, limits changes that require an incremental route using Rapid Recompile, and not a full compile.

Although having a Quartus Prime project is not required when using an `.stp`, it is recommended. The project database contains information about the integrity of the current Signal Tap Logic Analyzer session. Without the project database, there is no way to verify that the current `.stp` file matches the `.sof` file in the device. If you have an `.stp` file that does not match the `.sof` file, the Signal Tap Logic Analyzer can capture incorrect data.

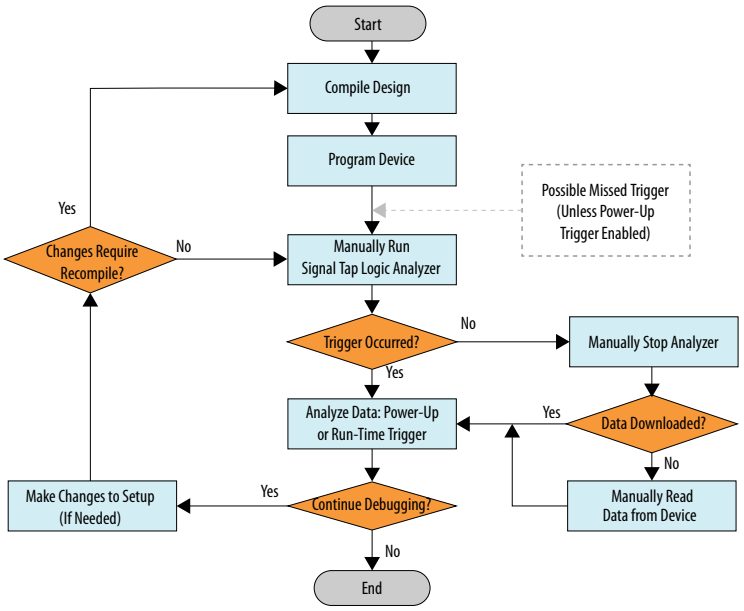
8.8 Run the Signal Tap Logic Analyzer

After the device is configured with your design that includes the Signal Tap Logic Analyzer, perform debugging operations in a manner similar to when you use an external logic analyzer. You initialize the logic analyzer by starting an analysis. When your trigger event occurs, the captured data is stored in the memory buffer on the device and then transferred to the `.stp` with the JTAG connection.

You can also perform the equivalent of a force trigger instruction that lets you view the captured data currently in the buffer without a trigger event occurring.

The figure illustrates a flow that shows how you operate the Signal Tap Logic Analyzer. The flowchart indicates where Power-Up and Runtime Trigger events occur and when captured data from these events is available for analysis.

Figure 180. Power-Up and Runtime Trigger Events Flowchart



You can also use In-System Sources and Probes in conjunction with the Signal Tap Logic Analyzer to force trigger conditions. The In-System Sources and Probes feature allows you to drive and sample values on to selected signals over the JTAG chain.

Related Links

Design Debugging with the Signal Tap Logic Analyzer on page 258

8.8.1 Runtime Reconfigurable Options

Certain settings in the `.stp` are changeable without recompiling your design when you use Runtime Trigger mode.

Table 96. Runtime Reconfigurable Features

Runtime Reconfigurable Setting	Description
Basic Trigger Conditions and Basic Storage Qualifier Conditions	All signals that have the Trigger condition turned on can be changed to any basic trigger condition value without recompiling.
Comparison Trigger Conditions and Comparison Storage Qualifier Conditions	All the comparison operands, the comparison numeric values, and the interval bound values are runtime-configurable. You can also switch from Comparison to Basic OR trigger at run-time without recompiling.
Advanced Trigger Conditions and Advanced Storage Qualifier Conditions	Many operators include runtime configurable settings. For example, all comparison operators are runtime-configurable. Configurable settings are shown with a white background in the block representation.This runtime reconfigurable option is turned on in the **Object Properties** dialog box.
Switching between a storage-qualified and a continuous acquisition	Within any storage-qualified mode, you can switch to continuous capture mode without recompiling the design. To enable this feature, turn on **disable storage qualifier**.
State-based trigger flow parameters	Refer to *Runtime Reconfigurable Settings, State-Based Triggering Flow*

Runtime Reconfigurable options can potentially save time during the debugging cycle by allowing you to cover a wider possible scenario of events without the need to recompile the design. You may experience a slight impact to the performance and logic utilization. You can turn off Runtime re-configurability for Advanced Trigger Conditions and the State-based trigger flow parameters, boosting performance and decreasing area utilization.

You can configure the `.stp` to prevent changes that normally require recompilation. To do this, in the **Setup** tab, select **Allow Trigger Condition changes only** above the node list.

Incremental Route lock mode, **Allow incremental route changes only**, limits changes which will only require an Incremental Route compilation, and not a full compile.

The example below illustrates a potential use case for Runtime Reconfigurable features. This example provides a storage qualified enabled State-based trigger flow description and shows how you can modify the size of a capture window at runtime without a recompile. This example gives you equivalent functionality to a segmented buffer with a single trigger condition where the segment sizes are runtime reconfigurable.

```
state ST1:
if ( condition1 && (c1 <= m) )    // each "segment"  triggers on condition
                                  //1
begin                               // m  = number of total "segments"
   start_store;
   increment c1;
   goto ST2:
End

else (c1 > m )                //This else condition handles the last
                              //segment.
begin
```

```
    start_store
    Trigger (n-1)
end

state ST2:
if ( c2 >= n)                //n = number of samples to capture in each
                             //segment.
begin
    reset c2;
    stop_store;
    goto ST1;
end

else (c2 < n)
begin
    increment c2;
    goto ST2;
end
```

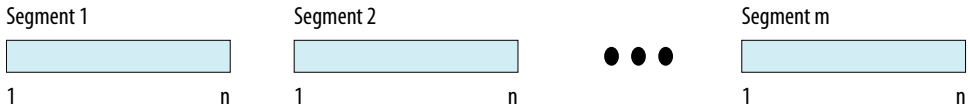
Note: *m* x *n* must equal the sample depth to efficiently use the space in the sample buffer.

The following figure shows a segmented buffer described by the trigger flow in example above.

During runtime, the values m and n are runtime reconfigurable. By changing the m and n values in the preceding trigger flow description, you can dynamically adjust the segment boundaries without incurring a recompile.

Figure 181. Segmented Buffer Created with Storage Qualifier and State-Based Trigger

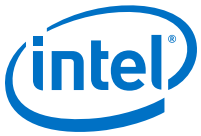
Total sample depth is fixed, where *m* x *n* must equal sample depth.



You can add states into the trigger flow description and selectively mask out specific states and enable other ones at runtime with status flags.

The example below shows a modified description of the example above with an additional state inserted. You use this extra state to specify a different trigger condition that does not use the storage qualifier feature. You insert status flags into the conditional statements to control the execution of the trigger flow.

```
state ST1 :
if (condition2  && f1)                      //additional state added for a non-
segmented
                                          //acquisition   Set f1 to enable state
begin
     start_store;
     trigger
end
else if (! f1)
     goto ST2;
state ST2:
if ( (condition1 && (c1 <= m)   && f2)   //f2  status flag used to mask state.
Set f2
                                         //to enable.
begin
   start_store;
   increment c1;
   goto ST3:
end
else (c1 > m )
```



```
   start_store
   Trigger (n-1)
end
state ST3:
if ( c2 >= n)
begin
    reset c2;
    stop_store;
    goto ST1;
end
else (c2 < n)
begin
    increment c2;
    goto ST2;
end
```

8.8.2 Signal Tap Status Messages

The table describes the text messages that might appear in the Signal Tap Status Indicator in the **Instance Manager** pane before, during, and after a data acquisition. Use these messages to monitor the state of the logic analyzer or what operation it is performing.

Table 97. Text Messages in the Signal Tap Status Indicator

Message	Message Description
`Not running`	The Signal Tap Logic Analyzer is not running. There is no connection to a device or the device is not configured.
`(Power-Up Trigger) Waiting for clock` (1)	The Signal Tap Logic Analyzer is performing a Runtime or Power-Up Trigger acquisition and is waiting for the clock signal to transition.
`Acquiring (Power-Up) pre-trigger data` (1)	The trigger condition has not been evaluated yet. A full buffer of data is collected if using the non-segmented buffer acquisition mode and storage qualifier type is continuous.
`Trigger In conditions met`	Trigger In condition has occurred. The Signal Tap Logic Analyzer is waiting for the condition of the first trigger condition to occur. This can appear if Trigger In is specified.
`Waiting for (Power-up) trigger` (1)	The Signal Tap Logic Analyzer is now waiting for the trigger event to occur.
`Trigger level <x>` met	The condition of trigger condition *x* has occurred. The Signal Tap Logic Analyzer is waiting for the condition specified in condition x + 1 to occur.
`Acquiring (power-up) post-trigger data` (1)	The entire trigger event has occurred. The Signal Tap Logic Analyzer is acquiring the post-trigger data. The amount of post-trigger data collected is you define between 12%, 50%, and 88% when the non-segmented buffer acquisition mode is selected.
`Offload acquired (Power-Up) data` (1)	Data is being transmitted to the Quartus Prime software through the JTAG chain.
`Ready to acquire`	The Signal Tap Logic Analyzer is waiting for you to initialize the analyzer.

1. This message can appear for both Runtime and Power-Up Trigger events. When referring to a Power-Up Trigger, the text in parentheses is added.

Note: In segmented acquisition mode, pre-trigger and post-trigger do not apply.

8.9 View, Analyze, and Use Captured Data

Use the Signal Tap Logic Analyzer interface to examine the data you captured manually or using a trigger, and use your findings to debug your design.

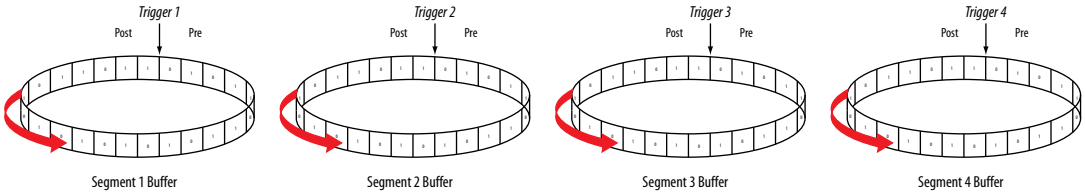
When in the Data view, you can use the drag-to-zoom feature by left-clicking to isolate the data of interest.

8.9.1 Capturing Data Using Segmented Buffers

Segmented Acquisition buffers allow you to perform multiple captures with a separate trigger condition for each acquisition segment. This feature allows you to capture a recurring event or sequence of events that span over a long period time efficiently.

Each acquisition segment acts as a non-segmented buffer, continuously capturing data when it is activated. When you run an analysis with the **segmented buffer** option enabled, the Signal Tap Logic Analyzer performs back-to-back data captures for each acquisition segment within your data buffer. The trigger flow, or the type and order in which the trigger conditions evaluate for each buffer, is defined by either the Sequential trigger flow control or the Custom State-based trigger flow control. The following figure shows a segmented acquisition buffer with four segments represented as four separate non-segmented buffers.

Figure 182. Segmented Acquisition Buffer



The Signal Tap Logic Analyzer finishes an acquisition with a segment, and advances to the next segment to start a new acquisition. Depending on when a trigger condition occurs, it may affect the way the data capture appears in the waveform viewer.The figure illustrates the method in which data is captured. The Trigger markers—Trigger 1, Trigger 2, Trigger 3 and Trigger 4—refer to the evaluation of the `segment_trigger` and `trigger` commands in the Custom State-based trigger flow. If you use a sequential flow, the Trigger markers refer to trigger conditions specified within the **Setup** tab.

If the Segment 1 Buffer is the active segment and Trigger 1 occurs, the Signal Tap Logic Analyzer starts evaluating Trigger 2 immediately. Data Acquisition for Segment 2 buffer starts when either Segment Buffer 1 finishes its post-fill count, or when Trigger 2 evaluates as TRUE, whichever condition occurs first. Thus, trigger conditions associated with the next buffer in the data capture sequence can preempt the post-fill count of the current active buffer. This allows the Signal Tap Logic Analyzer to accurately capture all of the trigger conditions that have occurred. Samples that have not been used appear as a blank space in the waveform viewer.

The next figure shows an example of a capture using sequential flow control with the trigger condition for each segment specified as **Don't Care**. Each segment before the last captures only one sample, because the next trigger condition immediately preempts capture of the current buffer. The trigger position for all segments is specified as pre-trigger (10% of the data is before the trigger condition and 90% of the data is after the trigger position). Because the last segment starts immediately with the trigger condition, the segment contains only post-trigger data. The three empty samples in the last segment are left over from the pre-trigger samples that the Signal Tap Logic Analyzer allocated to the buffer.

Figure 183. Segmented Capture with Preemption of Acquisition Segments

A segmented acquisition buffer using the sequential trigger flow with a trigger condition specified as Don't Care. All segments, with the exception of the last segment, capture only one sample because the next trigger condition preempts the current buffer from filling to completion.

For the sequential trigger flow, the **Trigger Position** option applies to every segment in the buffer. For maximum flexibility on how the trigger position is defined, use the custom state-based trigger flow. By adjusting the trigger position specific to your debugging requirements, you can help maximize the use of the allocated buffer space.

8.9.2 Differences in Pre-fill Write Behavior Between Different Acquisition Modes

The Signal Tap Logic Analyzer uses one of the following three modes when writing into the acquisition memory:

- **Non-segmented buffer**
- **Non-segmented buffer with a storage qualifier**
- **Segmented buffer**

There are subtle differences in the amount of data captured immediately after running the Signal Tap Logic Analyzer and before any trigger conditions occur. A non-segmented buffer, running in continuous mode, completely fills the buffer with sampled data before evaluating any trigger conditions. Thus, a non-segmented capture without any storage qualification enabled always shows a waveform with a full buffer's worth of data captured.

Filling the buffer provides you with as much data as possible within the capture window. The buffer gets pre-filled with data samples prior to evaluating the trigger condition. As such, Signal Tap requires that the buffer be filled at least once before any data can be retrieved through the JTAG connection and prevents the buffer from being dumped during the first acquisition prior to a trigger condition when you perform a **Stop Analysis**.

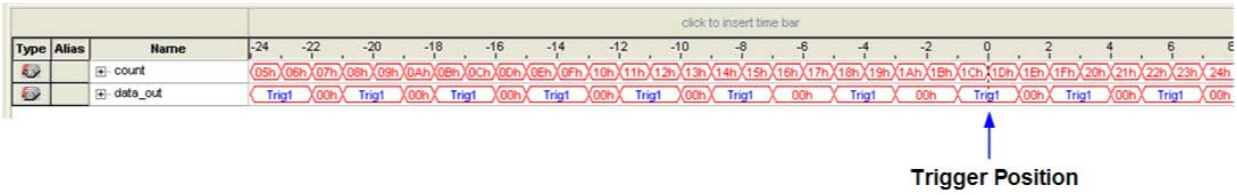
For segmented buffers and non-segmented buffers using any storage qualification mode, the Signal Tap Logic Analyzer immediately evaluates all trigger conditions while writing samples into the acquisition memory. The logic analyzer evaluates each trigger condition before acquiring a full buffer's worth of samples. This evaluation is especially important when using any storage qualification on the data set. The logic analyzer may miss a trigger condition if it waits until a full buffer's worth of data is captured before evaluating any trigger conditions.

If the trigger event occurs on any data sample before the specified amount of pre-trigger data has occurred, then the Signal Tap Logic Analyzer triggers and begins filling memory with post-trigger data, regardless of the amount of pre-trigger data you specify. For example, if you set the trigger position to 50% and set the logic analyzer to trigger on a processor reset, start the logic analyzer, and then power on your target system, the logic analyzer triggers. However, the logic analyzer memory is filled only

with post-trigger data, and not any pre-trigger data, because the trigger event, which has higher precedence than the capture of pre-trigger data, occurred before the pre-trigger condition was satisfied.

Figure 13–50 and Figure 13–51 on page 13–65 show the difference between a non-segmented buffer in continuous mode and a non-segmented buffer using a storage qualifier. The logic analyzer for the waveforms below is configured with a sample depth of 64 bits, with a trigger position specified as **Post trigger position**.

Figure 184. Signal Tap Logic Analyzer Continuous Data Capture

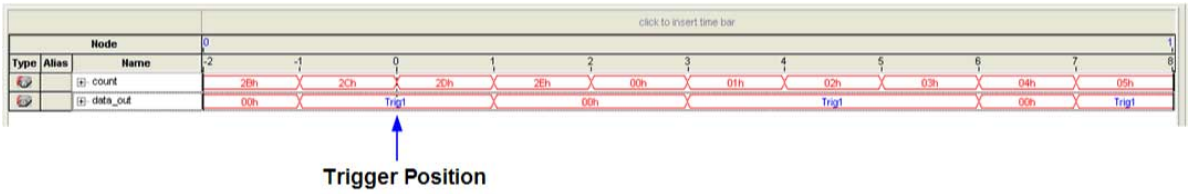


Note to **Figure 13–50** :

1. Continuous capture mode with post-trigger position.
2. Capture of a recurring pattern using a non-segmented buffer in continuous mode. The Signal Tap Logic Analyzer is configured with a basic trigger condition (shown in the figure as "Trig1") with a sample depth of 64 bits.

Notice in Figure 13–50 that Trig1 occurs several times in the data buffer before the Signal Tap Logic Analyzer actually triggers. A full buffer's worth of data is captured before the logic analyzer evaluates any trigger conditions. After the trigger condition occurs, the logic analyzer continues acquisition until it captures eight additional samples (12% of the buffer, as defined by the "post-trigger" position).

Figure 185. Signal Tap Logic Analyzer Conditional Data Capture



Note to **Figure 13–51** :

1. Conditional capture, storage always enabled, post-fill count.
2. Signal Tap Logic Analyzer capture of a recurring pattern using a non-segmented buffer in conditional mode. The logic analyzer is configured with a basic trigger condition (shown in the figure as "Trig1"), with a sample depth of 64 bits. The “Trigger in” condition is specified as "Don't care", which means that every sample is captured.

Notice in Figure 13–51 that the logic analyzer triggers immediately. As in Figure 13–50, the logic analyzer completes the acquisition with eight samples, or 12% of 64, the sample capacity of the acquisition buffer.

8.9.3 Creating Mnemonics for Bit Patterns

The mnemonic table feature allows you to assign a meaningful name to a set of bit patterns, such as a bus. To create a mnemonic table, right-click in the **Setup** or **Data** tab of an `.stp` and click **Mnemonic Table Setup**. You create a mnemonic table by entering sets of bit patterns and specifying a label to represent each pattern. Once you have created a mnemonic table, assign the table to a group of signals. To assign a mnemonic table, right-click on the group, click **Bus Display Format** and select the desired mnemonic table.

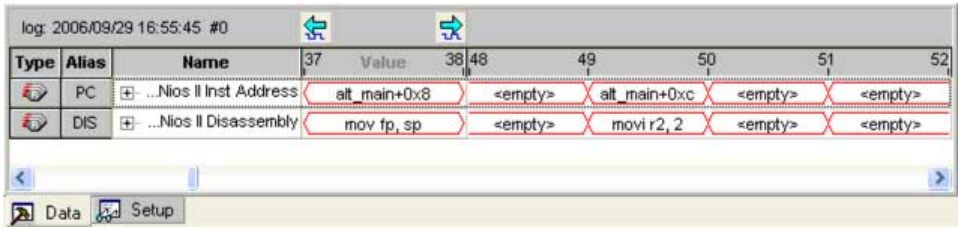
You use the labels you create in a table in different ways on the **Setup** and **Data** tabs. On the **Setup** tab, you can create basic triggers with meaningful names by right-clicking an entry in the **Trigger Conditions** column and selecting a label from the table you assigned to the signal group. On the **Data** tab, if any captured data matches a bit pattern contained in an assigned mnemonic table, the signal group data is replaced with the appropriate label, making it easy to see when expected data patterns occur.

8.9.4 Automatic Mnemonics with a Plug-In

When you use a plug-in to add signals to an `.stp`, mnemonic tables for the added signals are automatically created and assigned to the signals defined in the plug-in. To enable these mnemonic tables manually, right-click the name of the signal or signal group. On the **Bus Display Format** shortcut menu, then click the name of the mnemonic table that matches the plug-in.

As an example, the Nios II plug-in helps you to monitor signal activity for your design as the code is executed. If you set up the logic analyzer to trigger on a function name in your Nios II code based on data from an `.elf`, you can see the function name in the **Instance Address** signal group at the trigger sample, along with the corresponding disassembled code in the **Disassembly** signal group, as shown in Figure 13–52. Captured data samples around the trigger are referenced as offset addresses from the trigger function name.

Figure 186. Data Tab when the Nios II Plug-In is Used



8.9.5 Locating a Node in the Design

When you find the source of an error in your design using the Signal Tap Logic Analyzer, you can use the node locate feature to locate that signal in many of the tools found in the Quartus Prime software, as well as in your design files. This lets you find the source of the problem quickly so you can modify your design to correct the flaw. To locate a signal from the Signal Tap Logic Analyzer in one of the Quartus Prime software tools or your design files, right-click the signal in the `.stp`, and click **Locate in <tool name>**.

You can locate a signal from the node list with the following tools:

- Assignment Editor
- Pin Planner
- Timing Closure Floorplan
- Chip Planner
- Resource Property Editor
- Technology Map Viewer
- RTL Viewer
- Design File

8.9.6 Saving Captured Data

The data log shows the history of captured data and the triggers you used to capture the data. The Signal Tap Logic Analyzer acquires data, stores it in a log, and displays it as waveforms. When the logic analyzer is in auto-run mode and a trigger event occurs more than once, the Logic Analyzer stores captured data each time the trigger occurred as a separate entry in the data log. This allows you to review the captured data for each trigger event. The default name for a log is based on the time when the Logic Analyzer acquired the data. As a best practice, rename the data log with a more meaningful name.

The organization of logs is hierarchical; the Logic Analyzer groups similar logs of captured data in trigger sets. To open the **Data Log** pane, on the **View** menu, select **Data Log**. To turn on data logging, turn on **Enable data log** in the **Data Log**. To recall and activate a data log for a given trigger set, double-click the name of the data log in the list. The time stamping for the Data Log entries display the wall-clock time when Signal Tap triggered and the elapsed time from when acquisition started to when the device triggered.

Related Links

Manage Multiple Signal Tap Files and Configurations on page 281
You can use more than one `.stp` files in one design. Use the Data Log and the SOF Manager to handle multiple Signal Tap files and configurations.

8.9.7 Exporting Captured Data to Other File Formats

You can export captured data to the following file formats, for use with other EDA simulation tools:

- Comma Separated Values File (`.csv`)
- Table File (`.tbl`)
- Value Change Dump File (`.vcd`)
- Vector Waveform File (`.vwf`)
- Graphics format files (`.jpg`, `.bmp`)

To export the captured data from Signal Tap Logic Analyzer, on the File menu, click **Export** and specify the **File Name**, **Export Format**, and **Clock Period**.

8.9.8 Creating a Signal Tap List File

A `.stp` list file contains all the data the logic analyzer captures for a trigger event, in text format.

Each row of the list file corresponds to one captured sample in the buffer. Columns correspond to the value of each of the captured signals or signal groups for that sample. If you defined a mnemonic table for the captured data, a matching entry from the table replaces the numerical values in the list.

The `.stp` list file is especially useful when combined with a plug-in that includes instruction code disassembly. You can view the order of instruction code execution during the same time period of the trigger event.

To create a `.stp` list file in the Quartus Prime software, click **File ➤ Create/Update ➤ Create Signal Tap List File**.

Related Links

Adding Signals with a Plug-In on page 268
Instead of adding individual or grouped signals through the **Node Finder**, you can use a plug-in to add groups of relevant signals of a particular type of IP.

8.10 Other Features

The Signal Tap Logic Analyzer provides optional features not specific to a task flow. The following techniques may offer advantages in specific circumstances.

8.10.1 Debugging Partial Reconfiguration Designs Using Signal Tap Logic Analyzer

You can debug your PR design using Signal Tap. Quartus Prime software provides debug capabilities that allow you to acquire signals in static and PR regions simultaneously.
Moreover, you can debug multiple personas present in your PR region and multiple PR regions.

Note: The current version of Quartus Prime Pro Edition does not support debug of hierarchical PR regions using Signal Tap Logic Analyzer.

8.10.1.1 Recommendations when Debugging PR Designs

Follow these guidelines to obtain best results when debugging PR Designs with the Signal Tap Logic Analyzer:

- Include one `.stp` file per revision. For designs with multiple PR regions, you generate one revision for each PR region that you wish to debug.
- Tap pre-synthesis nodes only. In the Node Finder, filter by **Signal Tap: pre-synthesis**.
- Do not tap nodes in the default personas. Create a new PR implementation revision that instantiates the default persona, and tap nodes in the new revision.
- Store all the tapped nodes from a PR persona in one `.stp` file, to enable debugging the entire persona using only one Signal Tap window.
- Do not tap across PR regions, or from a static region to a PR region in the same `.stp` file.
- Each Signal Tap window opens only one `.stp` file. Therefore, to debug more than one partition simultaneously, open stand-alone Signal Tap windows from the command-line.

Related Links

- Set Up Partial Reconfiguration Design for Debug on page 321
  To debug a PR design, instantiate SLD JTAG bridges as part of the creation of the base revision, and define debug components for all PR personas. Optionally, you can specify signals to tap in the static region.
- Creating a Partial Reconfiguration Design
  In *Quartus Prime Pro Edition Handbook* Volume 1

8.10.1.2 Debug Infrastructure

During synthesis, the Compiler generates centralized debug managers—or hubs—for each region (static and PR) that contains debug agents. Each hub handles the debug agents in its partition. In the design hierarchy, the hub corresponding to the static region is `auto_fab_0`.

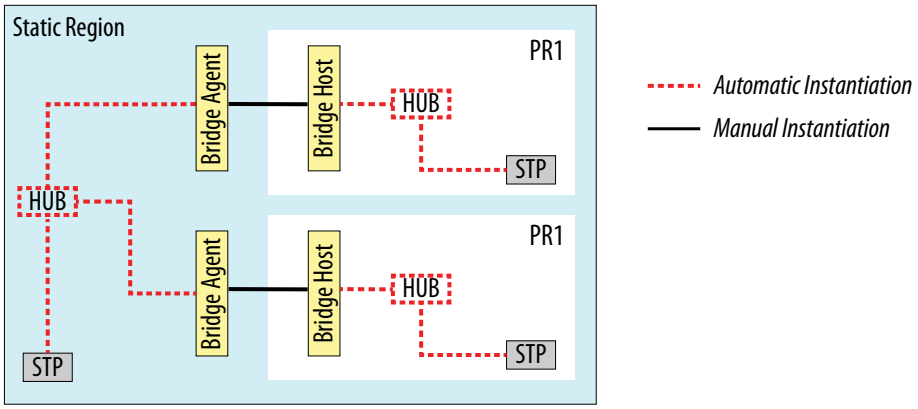
Manually connect the hubs on PR regions to the hub on the static region using the SLD JTAG bridge IP component. Using a bridge extends the parent partition SLD fabric into the child partition.

The SLD JTAG bridge consists of two IP components:

- **SLD JTAG Bridge Agent**—debug agent that resides in the static region.
- **SLD JTAG Bridge Host**—resides in the PR region. Connects to the PR JTAG hub on one end, and to the SLD JTAG bridge agent on the static region.

Figure 187. Debug Infrastructure in PR Design with Signal Tap

The figure shows in solid outline the entities that you instantiate manually, and in dashed outline the entities that synthesis instantiates.



For each PR region you debug, you need one instance of the SLD JTAG bridge agent in the static region and one instance of the SLD JTAG bridge host in the PR region.

During compilation, the synthesis engine performs the following functions:

- Generates a main JTAG hub in the static region.
- If the static region contains Signal Tap instances, connects those instances to the main JTAG hub.
- Detects bridge agent and bridge host instance(s).
- Connects the SLD JTAG bridge agent instances to the main JTAG hub.
- For each bridge host instance in a PR region that contains a Signal Tap instance:
  - Generates a PR JTAG hub in the PR region.
  - Connects all Signal Tap instances in the PR region to the PR JTAG hub.
  - Detects instance of the SLD JTAG bridge host.
  - Connects the PR JTAG hub to the JTAG bridge host.

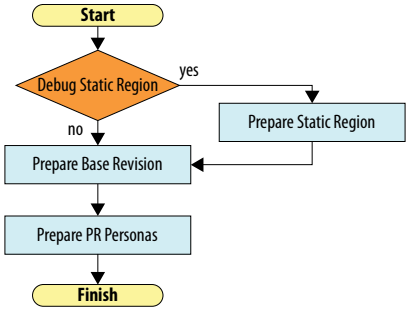
Related Links

System-Level Debugging Infrastructure on page 147
Intel FPGA on-chip debugging tools use the JTAG port to control and read-back data from debugging logic and signals under test.

8.10.1.3 Set Up Partial Reconfiguration Design for Debug

To debug a PR design, instantiate SLD JTAG bridges as part of the creation of the base revision, and define debug components for all PR personas. Optionally, you can specify signals to tap in the static region.

Figure 188. Setting Up PR Design for Debug with Signal Tap



After configuring all the PR personas in your design, continue the PR design flow.

Related Links

Partial Reconfiguration Design Flow
In *Quartus Prime Pro Edition Handbook* Volume 1

8.10.1.3.1 Prepare Static Region for Debug

To debug the static region in your PR design:

1. Tap nodes in the static region exclusively.
2. Save the `.stp` file. Use a name that identifies the file with the static region.
3. Enable Signal Tap in your project, and include `.stp` file in the base revision.

Note: Do not tap signals in the default PR personas.

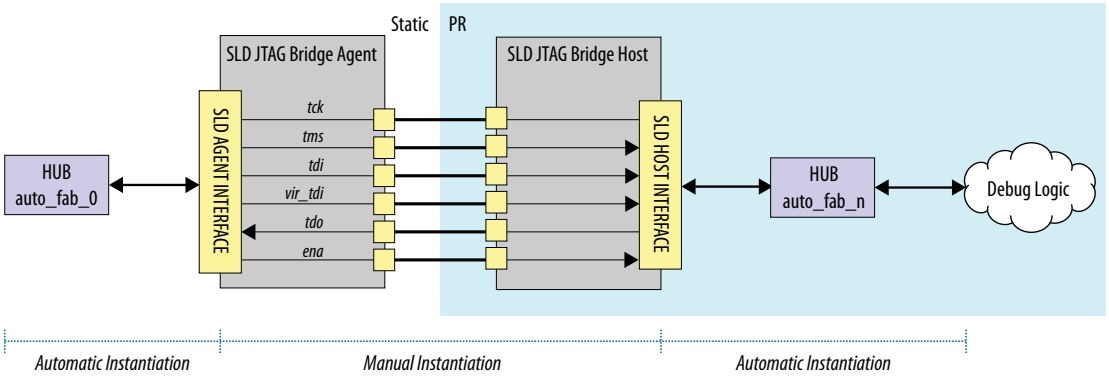
Related Links

Add Signals to the Signal Tap File on page 265
Add the signals that you want to monitor to the `.stp` node list. You can also select signals to define triggers.

8.10.1.3.2 Prepare Base Revision for Debug

Instantiate the SLD JTAG bridge components in your base revision using the IP Catalog or Qsys Pro. For each PR region in your design, instantiate one SLD JTAG bridge agent in the static region and one SLD JTAG bridge host in the default persona.

Figure 189. Signals in a SLD JTAG Bridge



Instantiate the SLD JTAG Bridge Agent

To instantiate the SLD JTAG bridge agent:

1. In the IP Catalog, navigate to **Library ➤ Basic Functions ➤ Simulation; Debug and Verification ➤ Debug and Performance**. Double click **SLD JTAG Agent**. Do not change the default values.
2. Specify a name for your bridge agent file and click **Generate HDL**.
3. Instantiate the SLD JTAG bridge agent in the static region.

Instantiate the SLD JTAG Bridge Host

To instantiate the SLD JTAG bridge host:

1. In the IP Catalog, navigate to **Library ➤ Basic Functions ➤ Simulation; Debug and Verification ➤ Debug and Performance**. Double-click **SLD JTAG Host**.
2. Specify a name for your SLD JTAG bridge host file and click **Generate HDL**.
3. Instantiate the SLD bridge host in the PR region.

8.10.1.3.3 Prepare PR Personas for Debug

Before you create revisions for personas in your design, you must instantiate debug IP components and tap signals.

For each PR persona that you wish to debug:

1. Instantiate the SLD JTAG bridge host in the PR persona.
2. Tap pre-synthesis nodes in the PR persona only.
3. Save in a new `.stp` file. Select a name that identifies the persona.
4. Use the new `.stp` file in the synthesis revision, and the same `.stp` file in the implementation revision for the persona.

If you don't wish to debug a particular persona, drive the `tdo` output signal to 0.

Related Links

- Instantiate the SLD JTAG Bridge Host on page 323
- Add Signals to the Signal Tap File on page 265
  Add the signals that you want to monitor to the `.stp` node list. You can also select signals to define triggers.

8.10.1.4 Data Acquisition

After generating the `.sof` and `.rbf` files for the revisions you wish to debug, you are ready to program your device and debug with the Signal Tap Logic Analyzer.

To perform data acquisition:

1. Program the base image into your device.
2. Partially reconfigure the device into the implementation you want to debug.
3. Open the Signal Tap Logic Analyzer by clicking **Tools ➤ Signal Tap Logic Analyzer** in the Quartus Prime software.

The Logic Analyzer opens and loads the `.stp` file set in the current active revision.

4. To debug other regions in your design, open new Signal Tap windows from the command-line:

```
quartus_stpw <stp_file.stp>
```

5. Debug your design with Signal Tap.

To debug another revision, you must partially reconfigure your design with the corresponding `.rbf` file.

Related Links

- Program the Target Device or Devices on page 309
- Run the Signal Tap Logic Analyzer on page 310
  After the device is configured with your design that includes the Signal Tap Logic Analyzer, perform debugging operations in a manner similar to when you use an external logic analyzer.
- View, Analyze, and Use Captured Data on page 313
  Use the Signal Tap Logic Analyzer interface to examine the data you captured manually or using a trigger, and use your findings to debug your design.

8.10.2 Creating Signal Tap File from Design Instance(s)

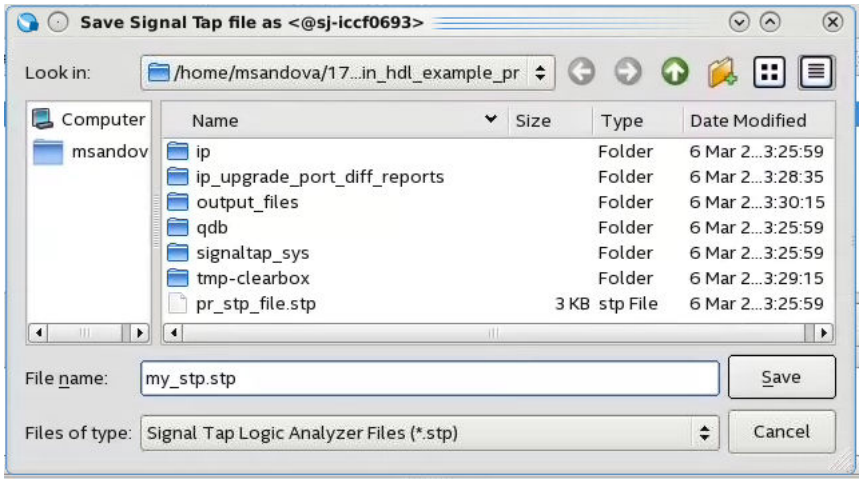
In addition to providing GUI support for generation of `.stp` files, the Quartus Prime software supports generation of a Signal Tap instance from logic defined in HDL source files. This technique is helpful to modify runtime configurable trigger conditions, acquire data, and view acquired data on the data log via Signal Tap utilities.

8.10.2.1 Creating a .stp File from a Design Instance

To generate a `.stp` file from parameterized HDL instance(s) within your design:

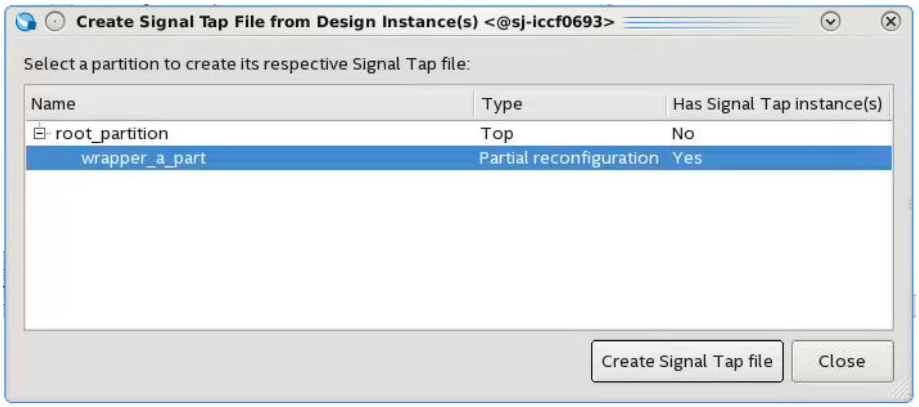
1. Open or create a Quartus Prime project that includes one or more HDL instances of the Signal Tap logic analyzer.
2. Click **Processing ➤ Start ➤ Start Analysis & Synthesis**.
3. Click **File ➤ Create/Update ➤ Create Signal Tap File from Design Instance(s)**.
4. Specify a location for the `.stp` file that generates, and click **Save**.

Figure 190. Create Signal Tap File from Design Instance(s) Dialog Box



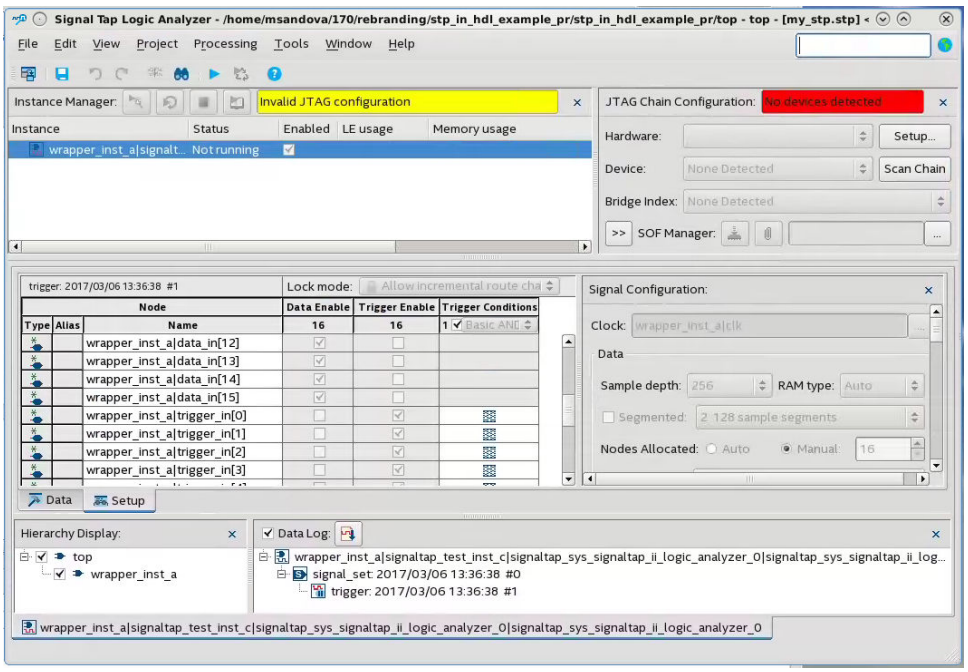
Note: If your project contains partial reconfiguration partitions, the **Create Signal Tap File from Design Instance(s)** dialog box displays a tree view of the PR partitions in the project. Select a partition from the view, and click **Create Signal Tap file**. The resultant `.stp` file that generates contains all HDL instances in the corresponding PR partition. The resultant `.stp` file does not include the instances in any nested partial reconfiguration partition.

Figure 191. Selecting Partition for `.stp` File Generation



After successful `.stp` file creation, the **Signal Tap Logic Analyzer** appears. All the fields are read-only, except runtime-configurable trigger conditions.

Figure 192. Generated `.stp` File



Related Links

- Signal Tap Command-Line Options on page 329
  To compile your design with the Signal Tap Logic Analyzer using the command prompt, use the `quartus_stp` command.
- Create Signal Tap File from Design Instance(s)
  in Quartus Prime Help
- Use the Parameter Editor to Create Your Logic Analyzer on page 264
  The parameter editor generates an HDL file that you instantiate in your design.
- Custom Trigger HDL Object on page 289
  The Custom Trigger HDL object allows you to create a custom trigger condition with your own HDL module in either Verilog or VHDL.

8.10.3 Using the Signal Tap MATLAB MEX Function to Capture Data

If you use MATLAB for DSP design, you can call the MATLAB MEX function `alt_signaltap_run`, built into the Quartus Prime software, to acquire data from the Signal Tap Logic Analyzer directly into a matrix in the MATLAB environment. If you use the MATLAB MEX function in a loop, you can perform as many acquisitions in the same amount of time as you can when using Signal Tap in the Quartus Prime software environment.

Note: The Signal Tap MATLAB MEX function is available in the Windows version and Linux version of the Quartus Prime software. It is compatible with MATLAB Release 14 Original Release Version 7 and later.

To set up the Quartus Prime software and the MATLAB environment to perform Signal Tap acquisitions, perform the following steps:

1. In the Quartus Prime software, create an **.stp** file.
2. In the node list in the **Data** tab of the Signal Tap Logic Analyzer Editor, organize the signals and groups of signals into the order in which you want them to appear in the MATLAB matrix. Each column of the imported matrix represents a single Signal Tap acquisition sample, while each row represents a signal or group of signals in the order they are organized in the **Data** tab.

   Note: Signal groups acquired from the Signal Tap Logic Analyzer and transferred into the MATLAB MEX function are limited to a width of 32 signals. If you want to use the MATLAB MEX function with a bus or signal group that contains more than 32 signals, split the group into smaller groups that do not exceed the 32-signal limit.
3. Save the **.stp** and compile your design. Program your device and run the Signal Tap Logic Analyzer to ensure your trigger conditions and signal acquisition work correctly.
4. In the MATLAB environment, add the Quartus Prime binary directory to your path with the following command:

   `addpath <Quartus install directory>\win`

   You can view the help file for the MEX function by entering the following command in MATLAB without any operators:

   `alt_signaltap_run`

   Use the MATLAB MEX function to open the JTAG connection to the device and run the Signal Tap Logic Analyzer to acquire data. When you finish acquiring data, close the JTAG connection.

   To open the JTAG connection and begin acquiring captured data directly into a MATLAB matrix called `stp`, use the following command:

```
stp = alt_signaltap_run \
('<stp filename>'[,('signed'|'unsigned')[,'<instance names>'[, \
'<signalset name>'[,'<trigger name>']]]]);
```

   When capturing data you must assign a filename, for example, *<stp filename>* as a requirement of the MATLAB MEX function. Other MATLAB MEX function options are described in Table 13–13.

Table 98. Signal Tap MATLAB MEX Function Options

Option	Usage	Description
`signed` `unsigned`	`'signed'` `'unsigned'`	The **signed** option turns signal group data into 32-bit two's-complement signed integers. The MSB of the group as defined in the Signal Tap **Data** tab is the sign bit. The **unsigned** option keeps the data as an unsigned integer. The default is **signed**.
<instance name>	`'auto_signaltap_0'`	Specify a Signal Tap instance if more than one instance is defined. The default is the first instance in the `.stp`, `auto_signaltap_0`.
<signal set name> *<trigger name>*	`'my_signalset'` `'my_trigger'`	Specify the signal set and trigger from the Signal Tap data log if multiple configurations are present in the `.stp`. The default is the active signal set and trigger in the file.

You can enable or disable verbose mode to see the status of the logic analyzer while it is acquiring data. To enable or disable verbose mode, use the following commands:

```
alt_signaltap_run('VERBOSE_ON');
alt_signaltap_run('VERBOSE_OFF');
```

When you finish acquiring data, close the JTAG connection with the following command:

```
alt_signaltap_run('END_CONNECTION');
```

For more information about the use of MATLAB MEX functions in MATLAB, refer to the MATLAB Help.

8.10.4 Using Signal Tap in a Lab Environment

You can install a stand-alone version of the Signal Tap Logic Analyzer. This version is particularly useful in a lab environment in which you do not have a workstation that meets the requirements for a complete Quartus Prime installation, or if you do not have a license for a full installation of the Quartus Prime software. The standalone version of the Signal Tap Logic Analyzer is included with and requires the Quartus Prime stand-alone Programmer which is available from the Downloads page of the Altera website.

8.10.5 Remote Debugging Using the Signal Tap Logic Analyzer

8.10.5.1 Debugging Using a Local PC and an SoC

You can use the System Console with Signal Tap Logic Analyzer to remote debug your Intel FPGA SoC. This method requires one local PC, an existing TCP/IP connection, a programming device at the remote location, and an Intel FPGA SoC.

Related Links

Remote Hardware Debugging over TCP/IP

8.10.5.2 Debugging Using a Local PC and a Remote PC

You can use the Signal Tap Logic Analyzer to debug a design that is running on a device attached to a PC in a remote location.

To perform a remote debugging session, you must have the following setup:

- The Quartus Prime software installed on the local PC
- Stand-alone Signal Tap Logic Analyzer or the full version of the Quartus Prime software installed on the remote PC
- Programming hardware connected to the device on the PCB at the remote location
- TCP/IP protocol connection

8.10.5.2.1 Equipment Setup

On the PC in the remote location, install the standalone version of the Signal Tap Logic Analyzer, included in the Quartus Prime standalone Programmer, or the full version of the Quartus Prime software. This remote computer must have Intel programming hardware connected, such as the or USB-Blaster.

On the local PC, install the full version of the Quartus Prime software. This local PC must be connected to the remote PC across a LAN with the TCP/IP protocol.

8.10.6 Using the Signal Tap Logic Analyzer in Devices with Configuration Bitstream Security

Certain device families support bitstream decryption during configuration using an on-device AES decryption engine. You can still use the Signal Tap Logic Analyzer to analyze functional data within the FPGA. However, note that JTAG configuration is not possible after the security key has been programmed into the device.

Intel FPGA recommends that you use an unencrypted bitstream during the prototype and debugging phases of the design. Using an unencrypted bitstream allows you to generate new programming files and reconfigure the device over the JTAG connection during the debugging cycle.

If you must use the Signal Tap Logic Analyzer with an encrypted bitstream, first configure the device with an encrypted configuration file using Passive Serial (PS), Fast Passive Parallel (FPP), or Active Serial (AS) configuration modes. The design must contain at least one instance of the Signal Tap Logic Analyzer. After the FPGA is configured with a Signal Tap Logic Analyzer instance in the design, when you open the Signal Tap Logic Analyzer in the Quartus Prime software, you then scan the chain and are ready to acquire data with the JTAG connection.

8.10.7 Backward Compatibility with Previous Versions of Quartus Prime Software

You can open an `.stp` created in a previous version in a current version of the Quartus Prime software. However, opening an `.stp` modifies it so that it cannot be opened in a previous version of the Quartus Prime software.

If you have a Quartus Prime project file from a previous version of the software, you may have to update the `.stp` configuration file to recompile the project. You can update the configuration file by opening the Signal Tap Logic Analyzer. If you need to update your configuration, a prompt appears asking if you would like to update the `.stp` to match the current version of the Quartus Prime software.

8.10.8 Signal Tap Command-Line Options

To compile your design with the Signal Tap Logic Analyzer using the command prompt, use the `quartus_stp` command. You can use the following options with the `quartus_stp` executable:

Table 99. `quartus_stp` Command-Line Options

Option	Usage	Description
`--stp_file` <*stp_filename*>	Mandatory	Specifies the name of the `.stp` file.
`--enable`	Optional	Creates assignments to the specified `.stp` in the `.qsf` and changes ENABLE_SIGNALTAP to ON. Includes Signal Tap Logic Analyzer in the next compilation. If no `.stp` is specified in the `.qsf`, the `--stp_file` option must be used. If omitted, the compiler uses the current value of ENABLE_SIGNALTAP in the `.qsf file` .
`--disable`	Optional	Removes the `.stp` reference from the `.qsf` and changes ENABLE_SIGNALTAP to OFF. The Signal Tap Logic Analyzer is removed from the design database the next time you compile your design. If the `--disable` option is omitted, the current value of ENABLE_SIGNALTAP in the `.qsf` is used.

The first example illustrates how to compile a design with the Signal Tap Logic Analyzer at the command line.

```
quartus_stp filtref --stp_file stp1.stp --enable
quartus_map filtref --source=filtref.bdf --family=CYCLONE
quartus_fit filtref --part=EP1C12Q240C6 --fmax=80MHz --tsu=8ns
quartus_asm filtref
```

The `quartus_stp --stp_file stp1.stp --enable` command creates the QSF variable and instructs the Quartus Prime software to compile the `stp1.stp` file with your design. The `--enable` option must be applied for the Signal Tap Logic Analyzer to compile into your design.

The example below shows how to create a new `.stp` after building the Signal Tap Logic Analyzer instance with the IP Catalog.

```
quartus_stp filtref --create_signaltap_hdl_file --stp_file stp1.stp
```

Related Links

Command-Line Scripting
In *Quartus Prime Pro Edition Handbook Volume 2*

8.10.9 Signal Tap Tcl Commands

The `quartus_stp` executable supports a Tcl interface that allows you to capture data without running the Quartus Prime GUI. You cannot execute Signal Tap Tcl commands from within the Tcl console in the Quartus Prime software. They must be executed from the command-line with the `quartus_stp` executable. To execute a Tcl file that has Signal Tap Logic Analyzer Tcl commands, use the following command:

```
quartus_stp -t <Tcl file>
```

The example is an excerpt from a script you can use to continuously capture data. Once the trigger condition is met, the data is captured and stored in the data log.

```
# Open Signal Tap session
open_session -name stp1.stp
# Start acquisition of instance auto_signaltap_0 and
#auto_signaltap_1 at the same time
# Calling run_multiple_end will start all instances
# run after run_multiple_start call
run_multiple_start
run -instance auto_signaltap_0 -signal_set signal_set_1 -trigger \
trigger_1 -data_log log_1 -timeout 5
run -instance auto_signaltap_1 -signal_set signal_set_1 -trigger \
trigger_1 -data_log log_1 -timeout 5
run_multiple_end
# Close Signal Tap session
close_session
```

When the script is completed, open the `.stp` that you used to capture data to examine the contents of the Data Log.

Related Links

::quartus::stp
In Quartus Prime Help

8.11 Design Example: Using Signal Tap Logic Analyzers

The system in this example contains many components, including a Nios processor, a direct memory access (DMA) controller, on-chip memory, and an interface to external SDRAM memory. After you press a button, the processor initiates a DMA transfer, which you analyze using the Signal Tap Logic Analyzer. In this example, the Nios processor executes a simple C program from on-chip memory and waits for you to press a button.

Related Links

AN 446: Debugging Nios II Systems with the Signal Tap Embedded Logic Analyzer application note

8.12 Custom Triggering Flow Application Examples

The custom triggering flow in the Signal Tap Logic Analyzer is most useful for organizing a number of triggering conditions and for precise control over the acquisition buffer. This section provides two application examples for defining a custom triggering flow within the Signal Tap Logic Analyzer. Both examples can be easily copied and pasted directly into the state machine description box by using the state display mode **All states in one window**.

Related Links

On-chip Debugging Design Examples website

8.12.1 Design Example 1: Specifying a Custom Trigger Position

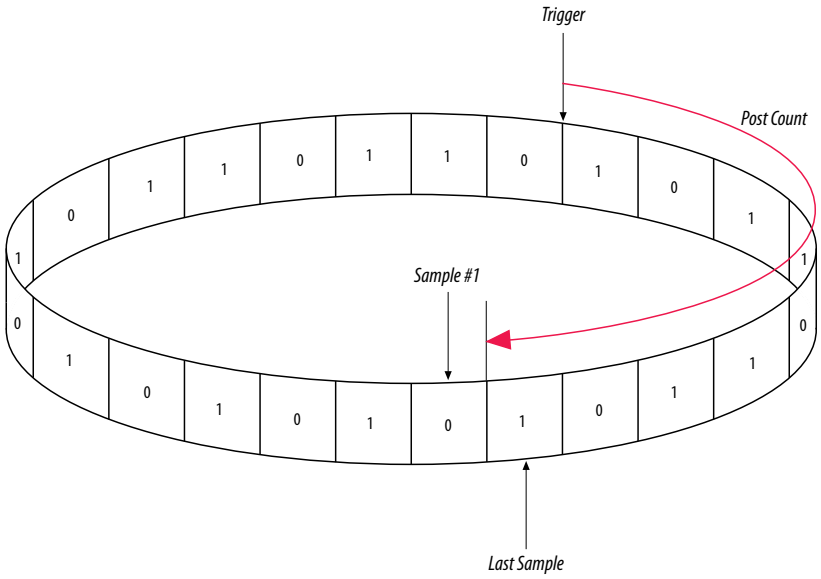
Actions to the acquisition buffer can accept an optional post-count argument. This post-count argument enables you to define a custom triggering position for each segment in the acquisition buffer. The example shows how to apply a trigger position

to all segments in the acquisition buffer. The example describes a triggering flow for an acquisition buffer split into four segments. If each acquisition segment is 64 samples in depth, the trigger position for each buffer will be at sample #34. The acquisition stops after all four segments are filled once.

```
if (c1 == 3 && condition1)
    trigger 30;
else if ( condition1 )
begin
    segment_trigger 30;
    increment c1;
end
```

Each segment acts as a non-segmented buffer that continuously updates the memory contents with the signal values. The last acquisition before stopping the buffer is displayed on the **Data** tab as the last sample number in the affected segment. The trigger position in the affected segment is then defined by *N* – `post count fill`, where N is the number of samples per segment. Figure 13–53 illustrates the triggering position.

Figure 193. Specifying a Custom Trigger Position



8.12.2 Design Example 2: Trigger When triggercond1 Occurs Ten Times between triggercond2 and triggercond3

The custom trigger flow description is often useful to count a sequence of events before triggering the acquisition buffer. The example shows such a sample flow. This example uses three basic triggering conditions configured in the Signal Tap **Setup** tab.

This example triggers the acquisition buffer when `condition1` occurs after `condition3` and occurs ten times prior to `condition3`. If `condition3` occurs prior to ten repetitions of `condition1`, the state machine transitions to a permanent wait state.

```
state ST1:
if ( condition2  )
begin
   reset c1;
   goto ST2;
end
State ST2 :
if ( condition1 )
    increment c1;
else if (condition3 && c1 < 10)
    goto ST3;
else if ( condition3 && c1 >= 10)
    trigger;
ST3:
goto ST3;
```

8.13 Signal Tap Scripting Support

You can run procedures and make settings described in this chapter in a Tcl script. You can also run some procedures at a command prompt. For detailed information about scripting command options, refer to the Quartus Prime Command-Line and Tcl API Help browser. To run the Help browser, type the following at the command prompt:

```
quartus_sh --qhelp
```

Related Links

Tcl Scripting
In *Quartus Prime Pro Edition Handbook Volume 2*

8.14 Document Revision History

Table 100. Document Revision History

Date	Version	Changes Made
2017.05.08	17.0.0	• Added: Open Standalone Signal Tap Logic Analyzer GUI. • Added: Debugging Partial Reconfiguration Designs Using Signal Tap Logic Analyzer. • Updated figures on Create Signal Tap File from Design Instance(s).
2016.10.31	16.1.0	• Implemented Intel rebranding. • Added: Create Signal Tap File from Design Instance(s). • Removed reference to unsupported Talkback feature.
2016.05.03	16.0.0	• Added: Specifying the Pipeline Factor • Added: Comparison Trigger Conditions
2015.11.02	15.1.0	• Changed instances of *Quartus II* to *Quartus Prime*. • Updated content to reflect Signal Tap support in Quartus Prime Pro Edition

continued...

Date	Version	Changes Made
2015.05.04	15.0.0	Added content for Floating Point Display Format in table: Signal Tap Logic Analyzer Features and Benefits.
2014.12.15	14.1.0	Updated location of Fitter Settings, Analysis & Synthesis Settings, and Physical Synthesis Optimizations to Compiler Settings.
December 2014	14.1.0	• Added MAX 10 as supported device. • Removed Full Incremental Compilation setting and Post-Fit (Strict) netlist type setting information. • Removed outdated GUI images from "Using Incremental Compilation with the Signal Tap Logic Analyzer" section.
June 2014	14.0.0	• DITA conversion. • Replaced MegaWizard Plug-In Manager and Megafunction content with IP Catalog and parameter editor content. • Added flows for custom trigger HDL object, Incremental Route with Rapid Recompile, and nested groups with Basic OR. • GUI changes: toolbar, drag to zoom, disable/enable instance, trigger log time-stamping.
November 2013	13.1.0	Removed HardCopy material. Added section on using cross-triggering with DS-5 tool and added link to white paper 01198. Added section on remote debugging an Altera SoC and added link to application note 693. Updated support for MEX function.
May 2013	13.0.0	• Added recommendation to use the state-based flow for segmented buffers with separate trigger conditions, information about Basic OR trigger condition, and hard processor system (HPS) external triggers. • Updated “Segmented Buffer” on page 13-17, Conditional Mode on page 13-21, Creating Basic Trigger Conditions on page 13-16, and Using External Triggers on page 13-48.
June 2012	12.0.0	Updated Figure 13–5 on page 13–16 and “Adding Signals to the Signal Tap File” on page 13–10.
November 2011	11.0.1	Template update. Minor editorial updates.
May 2011	11.0.0	Updated the requirement for the standalone Signal Tap software.
December 2010	10.0.1	Changed to new document template.
July 2010	10.0.0	• Add new acquisition buffer content to the “View, Analyze, and Use Captured Data” section. • Added script sample for generating hexadecimal CRC values in programmed devices. • Created cross references to Quartus Prime Help for duplicated procedural content.
November 2009	9.1.0	No change to content.
March 2009	9.0.0	• Updated Table 13–1 • Updated “Using Incremental Compilation with the Signal Tap Logic Analyzer” on page 13–45 • Added new Figure 13–33 • Made minor editorial updates
November 2008	8.1.0	Updated for the Quartus Prime software version 8.1 release:

continued...

Date	Version	Changes Made
		• Added new section "Using the Storage Qualifier Feature" on page 14–25 • Added description of `start_store` and `stop_store` commands in section "Trigger Condition Flow Control" on page 14–36 • Added new section "Runtime Reconfigurable Options" on page 14–63
May 2008	8.0.0	Updated for the Quartus Prime software version 8.0: • Added "Debugging Finite State machines" on page 14-24 • Documented various GUI usability enhancements, including improvements to the resource estimator, the bus find feature, and the dynamic display updates to the counter and flag resources in the State-based trigger flow control tab • Added "Capturing Data Using Segmented Buffers" on page 14–16 • Added hyperlinks to referenced documents throughout the chapter • Minor editorial updates

Related Links

Altera Documentation Archive

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

10 Debugging Single Event Upset Using the Fault Injection Debugger

You can detect and debug single event upset (SEU) using the Fault Injection Debugger in the Quartus Prime software. Use the debugger with the Altera Fault Injection IP core to inject errors into the configuration RAM (CRAM) of an FPGA device.

The injected error simulate the soft errors that can occur during normal operation due to (SEUs). Because SEUs are rare events, and therefore difficult to test, you can use the Fault Injection Debugger to induce intentional errors in the FPGA to test the system's response to these errors.

The Fault Injection Debugger is available for Stratix V family devices. For assistance with support for Arria V or Cyclone V family devices, file a service request using mySupport.

The Fault Injection Debugger provides the following benefits:

- Allows you to evaluate system response for mitigating single event functional interrupts (SEFI).
- Allows you to perform SEFI characterization, eliminating the need for entire system beam testing. Instead, you can limit the beam testing to failures in time (FIT)/Mb measurement at the device level.
- Scale FIT rates according to the SEFI characterization that is relevant to your design architecture. You can randomly distribute fault injections throughout the entire device, or constrain them to specific functional areas to speed up testing.
- Optimize your design to reduce SEU-caused disruption.

Related Links

Altera Website: Single Event Upsets

10.1 Single Event Upset Mitigation

Integrated circuits and programmable logic devices such as FPGAs are susceptible to SEUs. SEUs are random, nondestructive events, caused by two major sources: alpha particles and neutrons from cosmic rays. Radiation can cause either the logic register, embedded memory bit, or a configuration RAM (CRAM) bit to flip its state, thus leading to unexpected device operation.

Arria V, Cyclone V, Stratix V and newer devices have the following CRAM capabilities:

- Error Detection Cyclical Redundance Checking (EDCRC)
- Automatic correction of an upset CRAM (scrubbing)
- Ability to create an upset CRAM condition (fault injection)

For more information about SEU mitigation in Intel FPGA devices, refer to the *SEU Mitigation* chapter in the respective device handbook.

Related Links

Altera Website: Single Event Upsets

10.2 Hardware and Software Requirements

The following hardware and software is required to use the Fault Injection Debugger:

- Quartus Prime software version 14.0 or later.
- FEATURE line in your Intel FPGA license that enables the Fault Injection IP core. For more information, contact your local Intel FPGA sales representative.
- Download cable (USB-Blaster, USB-Blaster II, , or II cable).
- Intel FPGA development kit or user designed board with a JTAG connection to the device under test.
- (Optional) FEATURE line in your Intel FPGA license that enables the Advanced SEU Detection IP core.

Related Links

Altera Website: Contact Us

10.3 Using the Fault Injection Debugger and Fault Injection IP Core

The Fault Injection Debugger works together with the Fault Injection IP core. First, you instantiate the IP core in your design, compile, and download the resulting configuration file into your device. Then, you run the Fault Injection Debugger from within the Quartus Prime software or from the command line to simulate soft errors.

The Fault Injection Debugger communicates with the Fault Injection IP core via the JTAG interface. You perform debugging using the Fault Injection Debugger in the Quartus Prime software or using the command-line interface.

- The Fault Injection Debugger allows you to operate fault injection experiments interactively or by batch commands, and allows you to specify the logical areas in your design for fault injections.
- The command-line interface is useful for running the debugger via a script.

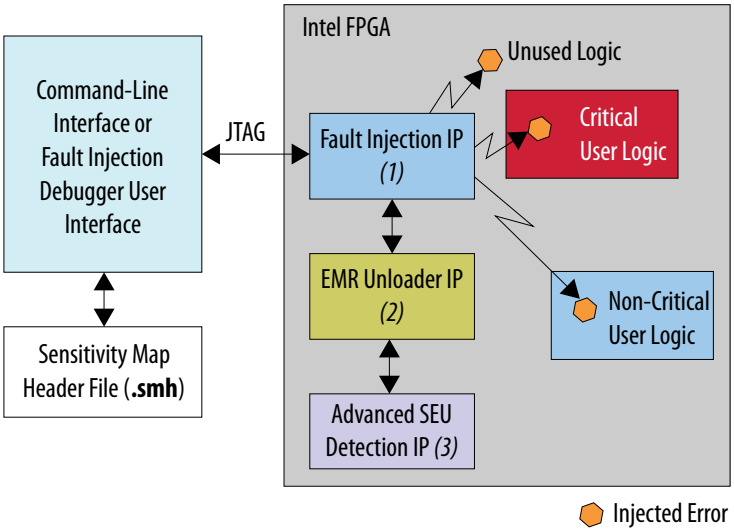
The Fault Injection IP accepts commands from the JTAG interface and reports status back through the JTAG interface.

Note: The Fault Injection IP core is implemented in soft logic in your device; therefore, you must account for this logic usage in your design. One methodology is to characterize your design's response to SEU in the lab and then omit the IP core from your final deployed design.

You use the Fault Injection IP core with the following IP cores:

- The Error Message Register (EMR) Unloader IP core, which reads and stores data from the hardened error detection circuitry in Intel FPGA devices.
- (Optional) The Advanced SEU Detection (ASD) IP core, which compares single-bit error locations to a sensitivity map during device operation to determine whether a soft error affects it.

Figure 194. Fault Injection Debugger Overview Block Diagram



Notes:
1. The fault Injection IP flips the bits of the targeted logic.
2. The Fault Injection Debugger and Advanced SEU Detection IP use the same EMR Unloader instance.
3. The Advanced SEU Detection IP core is optional.

Related Links

- Download Center
- AN 539: Test Methodology or Error Detection and Recovery using CRC in Intel FPGA Devices
- Understanding Single Event Functional Interrupts in FPGA Designs White Paper
- Altera Fault Injection IP Core User Guide
- Altera Error Message Unloader IP Core User Guide
- Altera Advanced SEU Detection (ALTERA_ADV_SEU_DETECTION) IP Core User Guide

10.3.1 Instantiating the Fault Injection IP Core

The Fault Injection IP core does not require you to set any parameters. To use the IP core, create a new IP instance, include it in your Qsys Pro system, and connect the signals as appropriate.

Note: You must use the Fault Injection IP core with the Error Message Register (EMR) Unloader IP core.

The Fault Injection and the EMR Unloader IP cores are available in Qsys Pro and the IP Catalog. Optionally, you can instantiate them directly into your RTL design, using Verilog HDL, SystemVerilog, or VHDL.

Related Links

Altera Fault Injection IP Core User Guide

10.3.1.1 Using the EMR Unloader IP Core

The EMR Unloader IP core provides an interface to the EMR, which is updated continuously by the device's EDCRC that checks the device's CRAM bits CRC for soft errors.

Figure 195. Example Qsys Pro System Including the Fault Injection IP Core and EMR Unloader IP Core

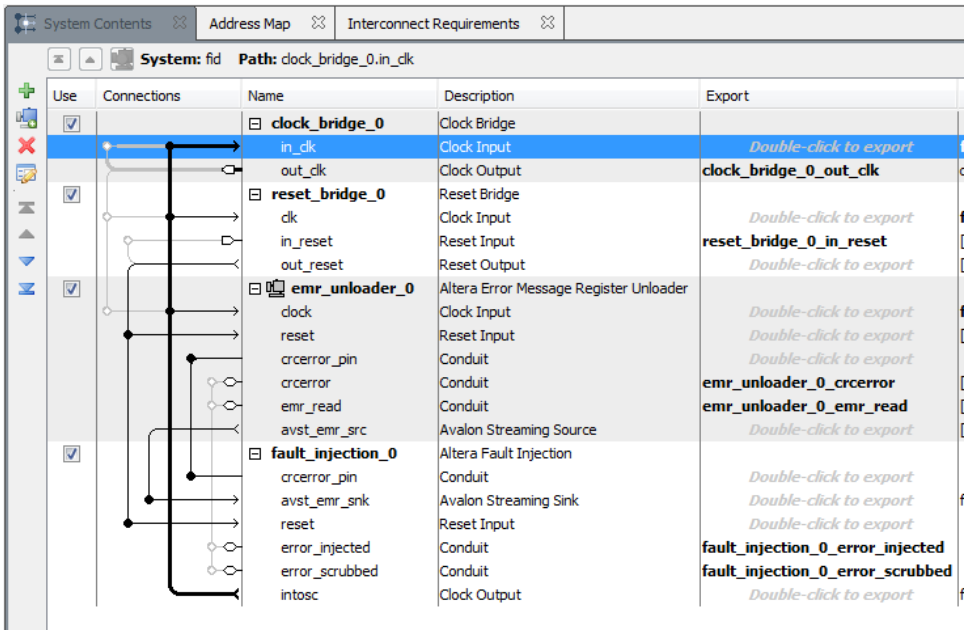
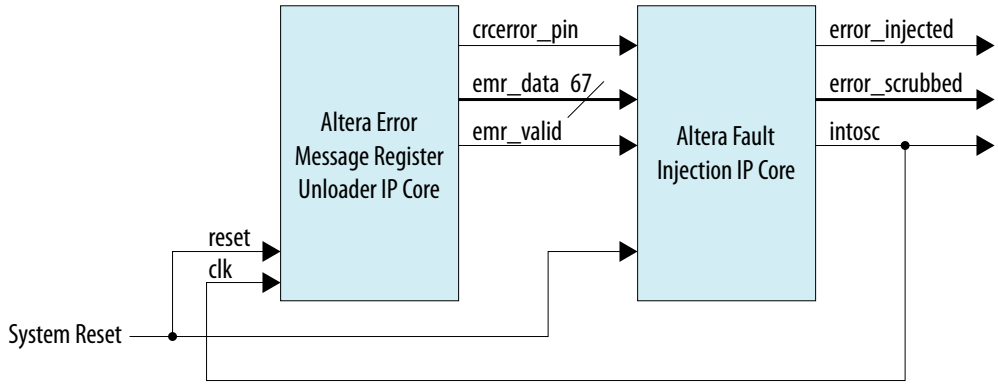


Figure 196. Example Altera Fault Injection IP Core and EMR Unloader IP Core Block Diagram



Related Links

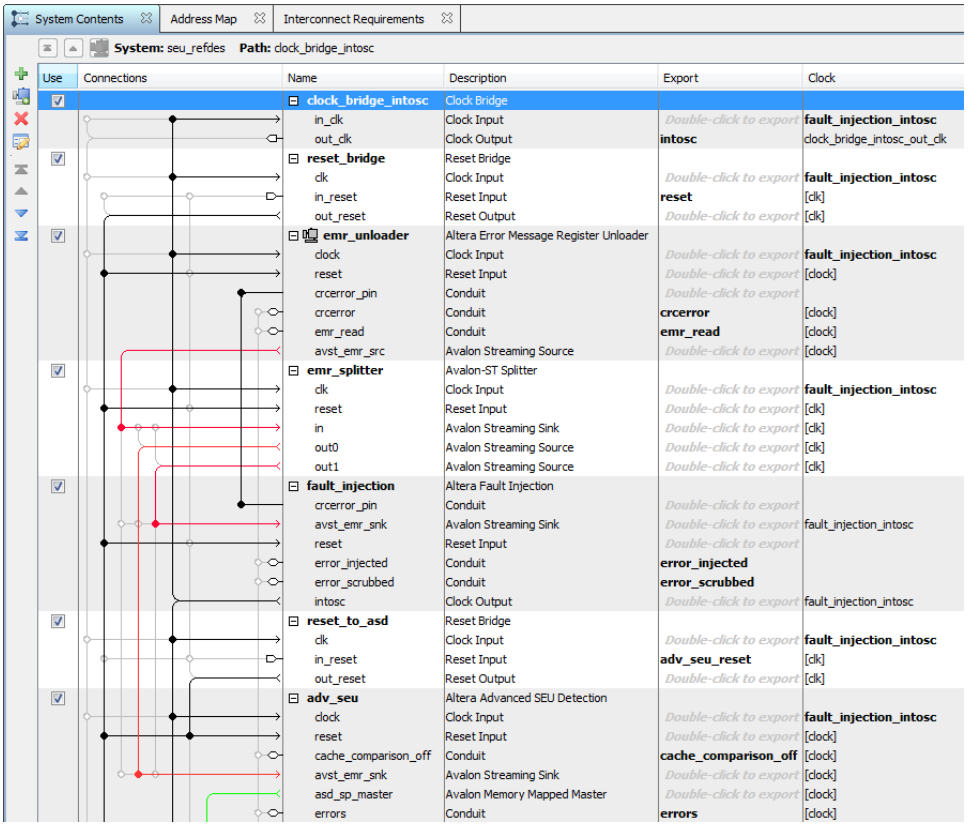
Altera Error Message Unloader IP Core User Guide

10.3.1.2 Using the Advanced SEU Detection IP Core

Use the Advanced SEU Detection (ASD) IP core when SEU tolerance is a design concern.

You must use the EMR Unloader IP core with the ASD IP core. Therefore, if you use the ASD IP and the Fault Injection IP in the same design, they must share the EMR Unloader output via an Avalon-ST splitter component. The following figure shows a Qsys Pro system in which an Avalon-ST splitter distributes the EMR contents to the ASD and Fault Injection IP cores.

Figure 197. Using the ASD and Fault Injection IP in the Same Qsys Pro System



Related Links

Altera Advanced SEU Detection (ALTERA_ADV_SEU_DETECTION) IP Core User Guide

10.3.2 Defining Fault Injection Areas

You can define specific regions of the FPGA for fault injection using a Sensitivity Map Header (**.smh**) file.

The SMH file stores the coordinates of the device CRAM bits, their assigned region (ASD Region), and criticality. During the design process you use hierarchy tagging to create the region. Then, during compilation, the Quartus Prime Assembler generates the SMH file. The Fault Injection Debugger limits error injections to specific device regions you define in the SMH file.

10.3.2.1 Performing Hierarchy Tagging

You define the FPGA regions for testing by assigning an ASD Region to the location. You can specify an ASD Region value for any portion of your design hierarchy using the Design Partitions Window.

1. Choose **Assignments ➤ Design Partitions Window**.
2. Right-click anywhere in the header row and turn on **ASD Region** to display the **ASD Region** column (if it is not already displayed).
3. Enter a value from 0 to 16 for any partition to assign it to a specific ASD Region.
   - — ASD region 0 is reserved to unused portions of the device. You can assign a partition to this region to specify it as non-critical..
   - — ASD region 1 is the default region. All used portions of the device are assigned to this region unless you explicitly change the ASD Region assignment.

10.3.2.2 About SMH Files

The SMH file contains the following information:

- If you are not using hierarchy tagging (i.e., the design has no explicit ASD Region assignments in the design hierarchy), the SMH file lists every CRAM bit and indicates whether it is sensitive for the design.
- If you have performed hierarchy tagging and changed default ASD Region assignments, the SMH file lists every CRAM bit and it's assigned ASD region.

The Fault Injection Debugger can limit injections to one or more specified regions.

Note: To direct the Assembler to generate an SMH file:

- Choose **Assignments ➤ Device ➤ Device and Pin Options ➤ Error Detection CRC**.
- Turn on the **Generate SEU sensitivity map file (.smh)** option.

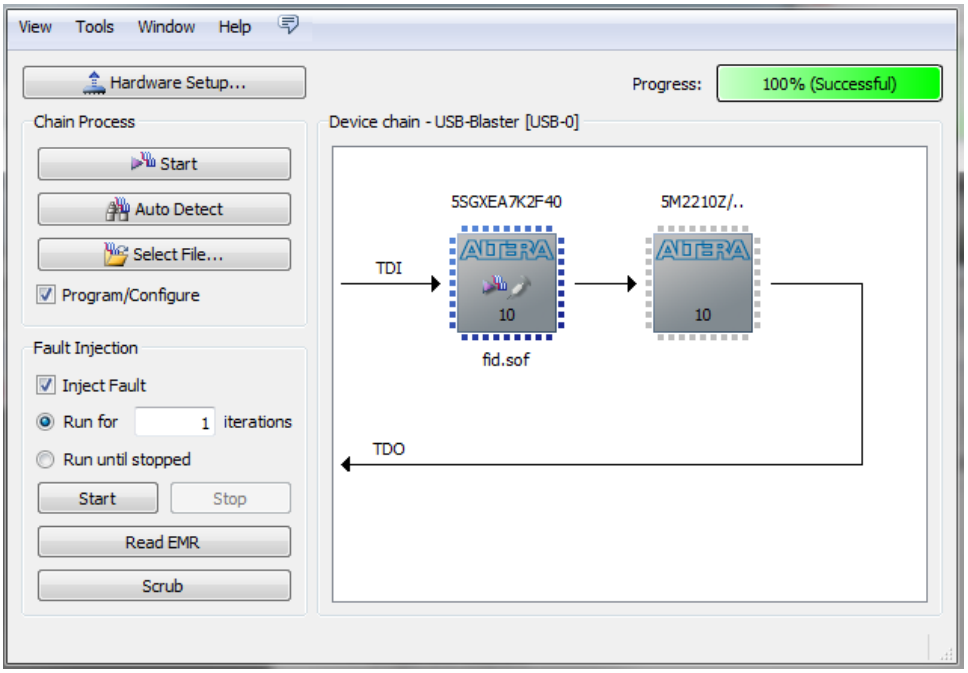
10.3.3 Using the Fault Injection Debugger

To use the Fault Injection Debugger, you connect from the tool to the device via the JTAG interface. Then, configure the device and perform fault injection.

To launch the Fault Injection Debugger, choose **Tools ➤ Fault Injection Debugger** in the Quartus Prime software.

Note: Configuring or programming the device is similar to the procedure used for the Programmer or Signal Tap Logic Analyzer.

Figure 198. Fault Injection Debugger



To configure your JTAG chain:

1. Click **Hardware Setup**. The tool displays the programming hardware connected to your computer.
2. Select the programming hardware you wish to use.
3. Click **Close**.
4. Click **Auto Detect**, which populates the device chain with the programmable devices found in the JTAG chain.

Related Links

Targeted Fault Injection Feature on page 348

10.3.3.1 Configuring Your Device and the Fault Injection Debugger

The Fault Injection Debugger uses a **.sof** and (optionally) a Sensitivity Map Header (**.smh**) file.

The Software Object File (**.sof**) configures the FPGA. The **.smh** file defines the sensitivity of the CRAM bits in the device. If you do not provide an **.smh** file, the Fault Injection Debugger injects faults randomly throughout the CRAM bits.

To specify a **.sof**:

1. Select the FPGA you wish to configure in the **Device chain** box.
2. Click **Select File**.
3. Navigate to the **.sof** and click **OK**. The Fault Injection Debugger reads the **.sof**.
4. (Optional) Select the SMH file.

If you do not specify an SMH file, the Fault Injection Debugger injects faults randomly across the entire device. If you specify an SMH file, you can restrict injections to the used areas of your device.

a. Right-click the device in the **Device chain** box and then click **Select SMH File**.

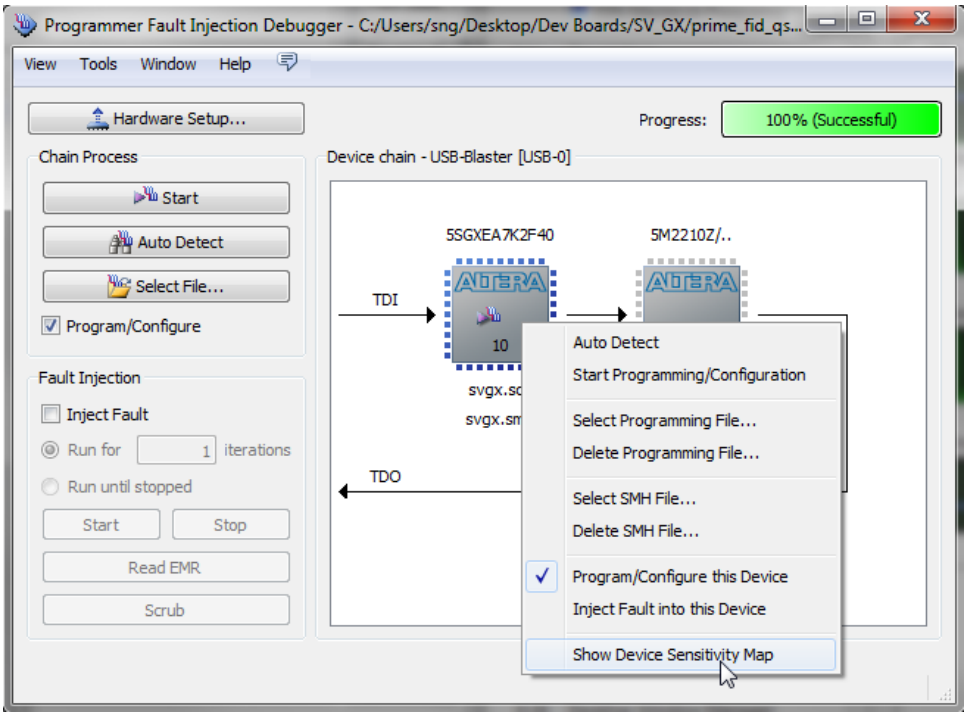
b. Select your SMH file.

c. Click **OK**.

5. Turn on **Program/Configure**.
6. Click **Start**.

The Fault Injection Debugger configures the device using the **.sof**.

Figure 199. Context Menu for Selecting the SMH File



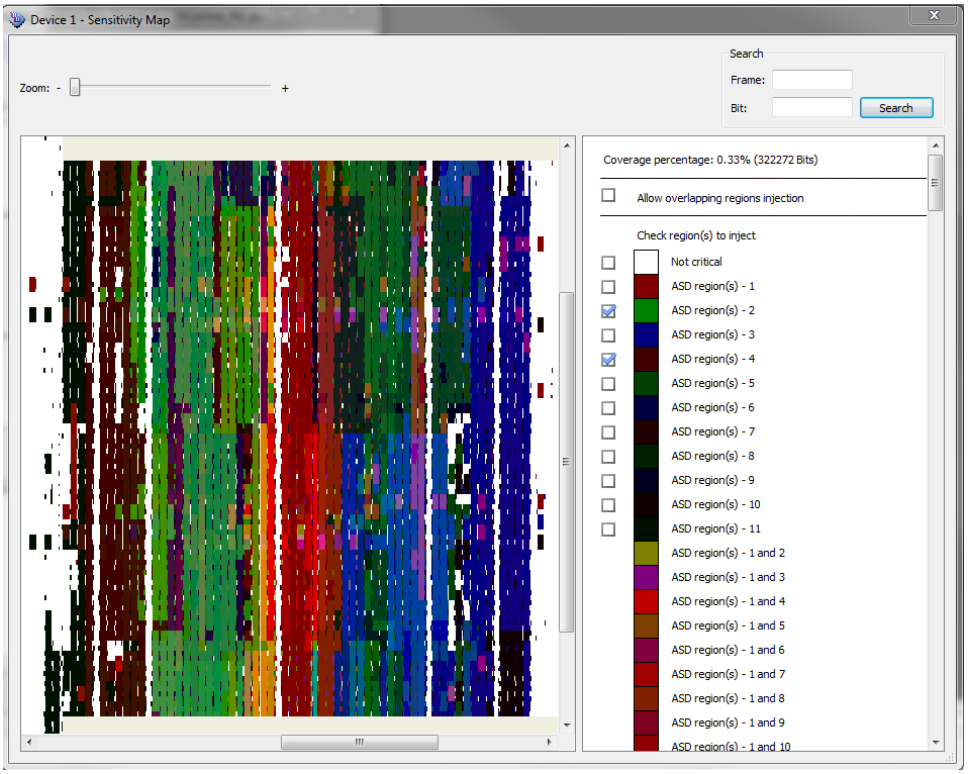
10.3.3.2 Constraining Regions for Fault Injection

After loading an SMH file, you can direct the Fault Injection Debugger to operate on only specific ASD regions.

To specify the ASD region(s) in which to inject faults:

1. Right-click the FPGA in the **Device chain** box and click Show Device Sensitivity Map.
2. Select the ASD region(s) for fault injection.

Figure 200. Sensitivity Map Viewer



10.3.3.3 Specifying Error Types

You can specify various types of errors for injection.

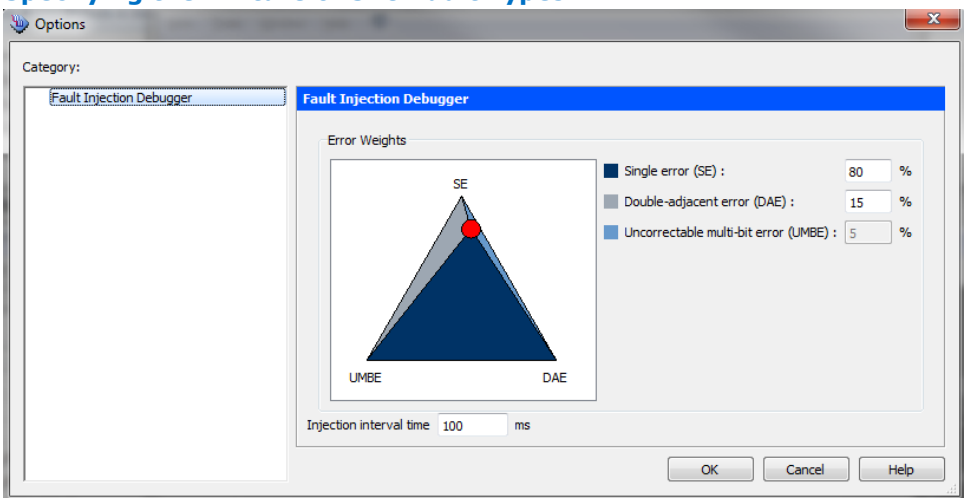
- Single errors (SE)
- Double-adjacent errors (DAE)
- Uncorrectable multi-bit errors (EMBE)

Intel FPGA devices can self-correct single and double-adjacent errors if the scrubbing feature is enabled. Intel FPGA devices cannot correct multi-bit errors. Refer to the chapter on mitigating SEUs for more information about debugging these errors.

You can specify the mixture of faults to inject and the injection time interval. To specify the injection time interval:

1. In the Fault Injection Debugger, choose **Tools ➤ Options**.
2. Drag the red controller to the mix of errors. Alternatively, you can specify the mix numerically.
3. Specify the **Injection interval time**.
4. Click **OK**.

Figure 201. Specifying the Mixture of SEU Fault Types



Related Links

Mitigating Single Event Upset

10.3.3.4 Injecting Errors

You can inject errors in several modes:

- Inject one error on command
- Inject multiple errors on command
- Inject errors until commanded to stop

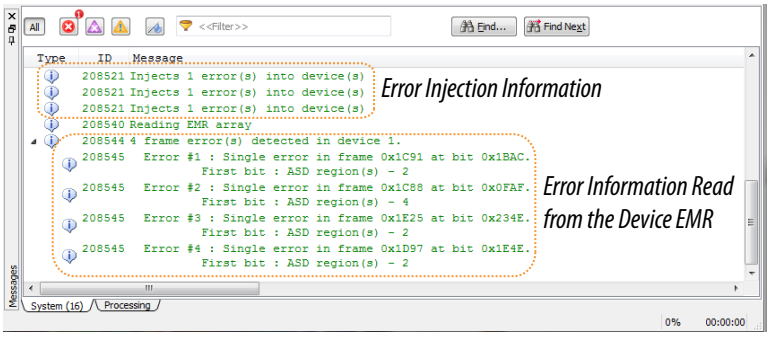
To inject these faults:

1. Turn on the **Inject Fault** option.
2. Choose whether you want to run error injection for a number of iterations or until stopped:
   - If you choose to run until stopped, the Fault Injection Debugger injects errors at the interval specified in the **Tools ➤ Options** dialog box.
   - If you want to run error injection for a specific number of iterations, enter the number.
3. Click **Start**.

   Note: The Fault Injection Debugger runs for the specified number of iterations or until stopped.

The Quartus Prime Messages window shows messages about the errors that are injected. For additional information on the injected faults, click **Read EMR**. The Fault Injection Debugger reads the device's EMR and displays the contents in the Messages window.

Figure 202. Quartus Prime Error Injection and EMR Content Messages



10.3.3.5 Recording Errors

You can record the location of any injected fault by noting the parameters reported in the Quartus Prime Messages window.

If, for example, an injected fault results in behavior you would like to replay, you can target that location for injection. You perform targeted injection using the Fault Injection Debugger command line interface.

10.3.3.6 Clearing Injected Errors

To restore the normal function of the FPGA, click **Scrub**. When you scrub an error, the device's EDCRC functions are used to correct the errors. The scrub mechanism is similar to that used during device operation.

10.3.4 Command-Line Interface

You can run the Fault Injection Debugger at the command line with the **quartus_fid** executable, which is useful if you want to perform fault injection from a script.

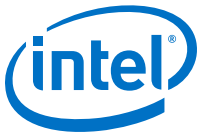
Table 101. Command line Arguments for Fault Injection

Short Argument	Long Argument	Description
c	cable	Specify programming hardware or cable. (Required)
i	index	Specify the active device to inject fault. (Required)
n	number	Specify the number of errors to inject. The default value is 1. (Optional)
t	time	Interval time between injections. (Optional)

Note: Use `quartus_fid --help` to view all available options.

The following code provides examples using the Fault Injection Debugger command-line interface.

```
##############################################
#
# Find out which USB cables are available for this instance
# The result shows that one cable is available, named "USB-Blaster"
#
$ quartus_fid --list
   . . .
```



```
    Info: Command: quartus_fid --list
    1) USB-Blaster on sj-sng-z4 [USB-0]
    Info: Quartus Prime 64-Bit Fault Injection Debugger was successful.
     0 errors, 0 warning
##############################################
#
# Find which devices are available on USB-Blaster cable
# The result shows two devices: a Stratix V A7, and a MAX V CPLD.
#
$ quartus_fid --cable USB-Blaster -a
    Info: Command: quartus_fid --cable=USB-Blaster -a
    Info (208809): Using programming cable "USB-Blaster on sj-sng-z4 [USB-0]"
    1) USB-Blaster on sj-sng-z4 [USB-0]
      029030DD   5SGXEA7H(1|2|3)/5SGXEA7K1/..
      020A40DD   5M2210Z/EPM2210
    Info: Quartus Prime 64-Bit Fault Injection Debugger was successful.
    0 errors, 0 warnings

##############################################
#
# Program the Stratix V device
# The --index option specifies operations performed on a connected device.
#   "=svgx.sof" associates a .sof file with the device
#   "#p" means program the device
#
$ quartus_fid --cable USB-Blaster --index "@1=svgx.sof#p"
 . . .
    Info (209016): Configuring device index 1
    Info (209017): Device 1 contains JTAG ID code 0x029030DD
    Info (209007): Configuration succeeded -- 1 device(s) configured
    Info (209011): Successfully performed operation(s)
    Info (208551): Program signature into device 1.
    Info: Quartus Prime 64-Bit Fault Injection Debugger was successful.
    0 errors, 0 warnings

##############################################
#
# Inject a fault into the device.
# The #i operator indicates to inject faults
# -n 3 indicates to inject 3 faults
#
$ quartus_fid --cable USB-Blaster --index "@1=svgx.sof#i" -n 3
    Info: Command: quartus_fid --cable=USB-Blaster --index=@1=svgx.sof#i -n 3
    Info (208809): Using programming cable "USB-Blaster on sj-sng-z4 [USB-0]"
    Info (208521): Injects 3 error(s) into device(s)
    Info: Quartus Prime 64-Bit Fault Injection Debugger was successful.
    0 errors, 0 warnings


##############################################
#
# Interactive Mode.
# Using the #i operation with -n 0 puts the debugger into interactive mode.
# Note that 3 faults were injected in the previous session;
# "E" reads the faults currently in the EMR Unloader IP core.
#
$ quartus_fid --cable USB-Blaster --index "@1=svgx.sof#i" -n 0
    Info: Command: quartus_fid --cable=USB-Blaster --index=@1=svgx.sof#i -n 0
    Info (208809): Using programming cable "USB-Blaster on sj-sng-z4 [USB-0]"
    Enter :
          'F' to inject fault
          'E' to read EMR
          'S' to scrub error(s)
          'Q' to quit
    E
    Info (208540): Reading EMR array
    Info (208544): 3 frame error(s) detected in device 1.
        Info (208545):   Error #1 : Single error in frame 0x1028 at bit 0x21EA.
        Info (10914):   Error #2 : Uncorrectable multi-bit error in frame 0x1116.
        Info (208545):   Error #3 : Single error in frame 0x1848 at bit 0x128C.
    Enter :
```

```
          'F' to inject fault
          'E' to read EMR
          'S' to scrub error(s)
          'Q' to quit
    Q
    Info: Quartus Prime 64-Bit Fault Injection Debugger was successful.
        0 errors, 0 warnings
        Info: Peak virtual memory: 1522 megabytes
        Info: Processing ended: Mon Nov  3 18:50:00 2014
        Info: Elapsed time: 00:00:29
        Info: Total CPU time (on all processors): 00:00:13
```

10.3.4.1 Targeted Fault Injection Feature

The Fault Injection Debugger injects faults into the FPGA randomly. However, the Targeted Fault Injection feature allows you to inject faults into targeted locations in the CRAM. This operation may be useful, for example, if you noted an SEU event and want to test the FPGA or system response to the same event after modifying a recovery strategy.

Note: The Targeted Fault Injection feature is available only from the command line interface.

You can specify that errors are injected from the command line or in prompt mode.

Related Links

AN 539: Test Methodology or Error Detection and Recovery using CRC in Intel FPGA Devices

10.3.4.1.1 Specifying an Error List From the Command Line

The Targeted Fault Injection feature allows you to specify an error list from the command line, as shown in the following example:

```
c:\Users\sng> quartus_fid -c 1 - i "@1= svgx.sof#i " -n 2 -user="@1=
0x2274 0x05EF 0x2264 0x0500"
```

Where:

`c 1` indicates that the fpga is controlled by the first cable on your computer.

`i "@1= svgx.sof#i "` indicates that the first device in the chain is loaded with the object file **svgx.sof** and will be injected with faults.

`n 2` indicates that two faults will be injected.

`user=”@1= 0x2274 0x05EF 0x2264 0x0500”` is a user-specified list of faults to be injected. In this example, device 1 has two faults: at frame 0x2274, bit 0x05EF and at frame 0x2264, bit 0x0500.

10.3.4.1.2 Specifying an Error List From Prompt Mode

You can operate the Targeted Fault Injection feature interactively by specifying the number of faults to be 0 (-n 0). The Fault Injection Debugger presents prompt mode commands and their descriptions.

Prompt Mode Command	Description
F	Inject a fault
E	Read the EMR
S	Scrub errors
Q	Quit

In prompt mode, you can issue the F command alone to inject a single fault in a random location in the device. In the following examples using the F command in prompt mode, three errors are injected.

```
F #3 0x12 0x34 0x56 0x78 * 0x9A 0xBC +
```

- Error 1 – Single bit error at frame 0x12, bit 0x34
- Error 2 – Uncorrectable error at frame 0x56, bit 0x78 (an * indicates a multi-bit error)
- Error 3 – Double-adjacent error at frame 0x9A, bit 0xBC (a + indicates a double bit error)

```
F 0x12 0x34 0x56 0x78 *
```

One (default) error is injected:

Error 1 – Single bit error at frame 0x12, bit 0x34. Locations after the first frame/bit location are ignored.

```
F #3 0x12 0x34 0x56 0x78 * 0x9A 0xBC + 0xDE 0x00
```

Three errors are injected:

- Error 1 – Single bit error at frame 0x12, bit 0x34
- Error 2 – Uncorrectable error at frame 0x56, bit 0x78
- Error 3 – Double-adjacent error at frame 0x9A, bit 0xBC
- Locations after the first 3 frame/bit pairs are ignored

10.3.4.1.3 Determining CRAM Bit Locations

When the Fault Injection Debugger detects a CRAM EDCRC error, the Error Message Register (EMR) contains the syndrome, frame number, bit location, and error type (single, double, or multi-bit) of the detected CRAM error.

During system testing, save the EMR contents reported by the Fault Injection Debugger when you detect an EDCRC fault.

Note: With the recorded EMR contents, you can supply the frame and bit numbers to the Fault Injection Debugger to replay the errors noted during system testing, to further design, and characterize a system recovery response to that error.

Related Links

AN 539: Test Methodology or Error Detection and Recovery using CRC in Intel FPGA Devices

10.3.4.2 Advanced Command-Line Options: ASD Regions and Error Type Weighting

You can use the Fault Injection Debugger command-line interface to inject errors into ASD regions and weight the error types.

First, you specify the mix of error types (single bit, double adjacent, and multi-bit uncorrectable) using the `--weight <single errors>.<double adjacent errors>.<multi-bit errors>` option. For example, for a mix of 50% single errors, 30% double adjacent errors, and 20% multi-bit uncorrectable errors, use the option `--weight=50.30.20`. Then, to target an ASD region, use the `-smh` option to include the SMH file and indicate the ASD region to target. For example:

```
$ quartus_fid --cable=USB-BlasterII --index "@1=svgx.sof#pi" --
weight=100.0.0 --smh="@1=svgx.smh#2" --number=30
```

This example command:

- Programs the device and injects faults (`pi` string)
- Injects 100% single-bit faults (100.0.0)
- Injects only into `ASD_REGION` 2 (indicated by the `#2`)
- Injects 30 faults

10.4 Document Revision History

Table 102. Document Revision History

Date	Version	Changes
2016.10.31	16.1.0	• Implemented Intel rebranding.
2015.11.02	15.1.0	Changed instances of *Quartus II* to *Quartus Prime*.
2015.05.04	15.0.0	• Provided more detail on how to use the Fault Injection Debugger throughout the document. • Added more command-line examples.
2014.06.30	14.0.0	• Removed "Modifying the Quartus INI File" section. • Added "Targeted Fault Injection Feature" section. • Updated "Hardware and Software Requirements" section.
December 2012	2012.12.01	Preliminary release.

11 In-System Debugging Using External Logic Analyzers

11.1 About the Quartus Prime Logic Analyzer Interface

The Quartus Prime Logic Analyzer Interface (LAI) allows you to use an external logic analyzer and a minimal number of Intel-supported device I/O pins to examine the behavior of internal signals while your design is running at full speed on your Intel-supported device.

The LAI connects a large set of internal device signals to a small number of output pins. You can connect these output pins to an external logic analyzer for debugging purposes. In the Quartus Prime LAI, the internal signals are grouped together, distributed to a user-configurable multiplexer, and then output to available I/O pins on your Intel-supported device. Instead of having a one-to-one relationship between internal signals and output pins, the Quartus Prime LAI enables you to map many internal signals to a smaller number of output pins. The exact number of internal signals that you can map to an output pin varies based on the multiplexer settings in the Quartus Prime LAI.

Note: The term "logic analyzer" when used in this document includes both logic analyzers and oscilloscopes equipped with digital channels, commonly referred to as mixed signal analyzers or MSOs.

The LAI does not support Hard Processor System (HPS) I/Os.

Related Links

Device Support Center

11.2 Choosing a Logic Analyzer

The Quartus Prime software offers the following two general purpose on-chip debugging tools for debugging a large set of RTL signals from your design:

- The Signal Tap Logic Analyzer
- An external logic analyzer, which connects to internal signals in your Intel-supported device by using the Quartus Prime LAI

Table 103. Comparing the Signal Tap Logic Analyzer with the Logic Analyzer Interface

Feature	Description	Recommended Logic Analyzer
Sample Depth	You have access to a wider sample depth with an external logic analyzer. In the Signal Tap Logic Analyzer, the maximum sample depth is set to	LAI

continued...

Feature	Description	Recommended Logic Analyzer
	128 Kb, which is a device constraint. However, with an external logic analyzer, there are no device constraints, providing you a wider sample depth.	
Debugging Timing Issues	Using an external logic analyzer provides you with access to a "timing" mode, which enables you to debug combined streams of data.	LAI
Performance	You frequently have limited routing resources available to place and route when you use the Signal Tap Logic Analyzer with your design. An external logic analyzer adds minimal logic, which removes resource limits on place-and-route.	LAI
Triggering Capability	The Signal Tap Logic Analyzer offers triggering capabilities that are comparable to external logic analyzers.	LAI or Signal Tap
Use of Output Pins	Using the Signal Tap Logic Analyzer, no additional output pins are required. Using an external logic analyzer requires the use of additional output pins.	Signal Tap
Acquisition Speed	With the Signal Tap Logic Analyzer, you can acquire data at speeds of over 200 MHz. You can achieve the same acquisition speeds with an external logic analyzer; however, you must consider signal integrity issues.	Signal Tap

Related Links

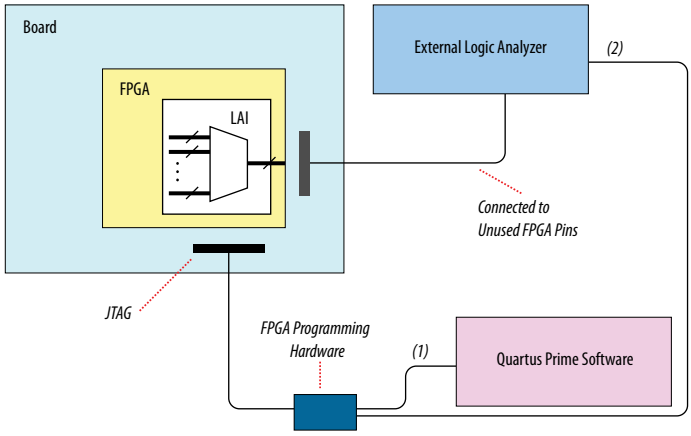
System Debugging Tools Overview on page 146

11.2.1 Required Components

To perform analysis using the LAI you need the following components:

- Quartus Prime software version 15.1 or later
- The device under test
- An external logic analyzer
- An Intel FPGA communications cable
- A cable to connect the Intel-supported device to the external logic analyzer

Figure 203. LAI and Hardware Setup

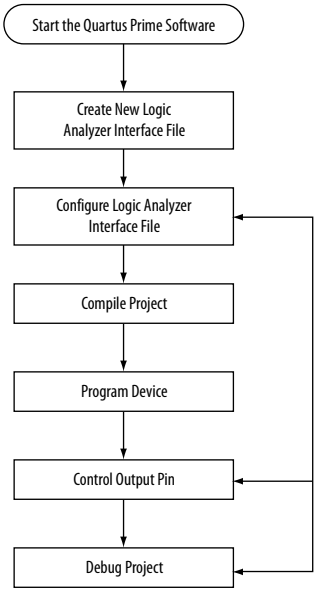


Notes to figure:

1. Configuration and control of the LAI using a computer loaded with the Quartus Prime software via the JTAG port.
2. Configuration and control of the LAI using a third-party vendor logic analyzer via the JTAG port. Support varies by vendor.

11.3 Flow for Using the LAI

Figure 204. LAI Workflow



Notes to figure:

1. Configuration and control of the LAI using a computer loaded with the Quartus Prime software via the JTAG port.
2. Configuration and control of the LAI using a third-party vendor logic analyzer via the JTAG port. Support varies by vendor.

11.4 Working with LAI Files

The `.lai` file stores the configuration of an LAI instance. The `.lai` file opens in the LAI editor. The editor allows you to group multiple internal signals to a set of external pins.

11.4.1 Configuring the File Core Parameters

After you create the `.lai` file, configure the `.lai` file core parameters by clicking on the **Setup View** list, and then selecting **Core Parameters**. The table below lists the `.lai` file core parameters.

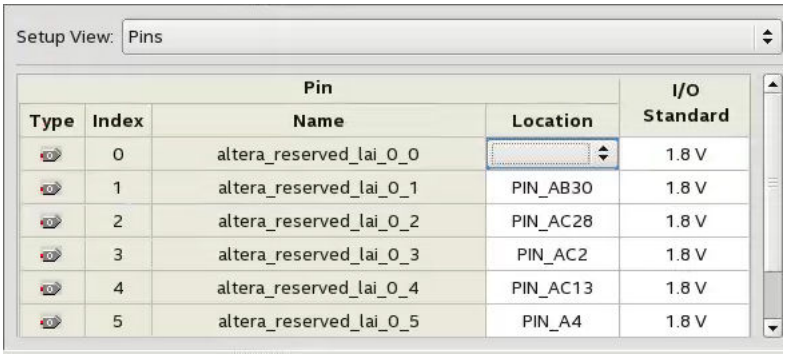
Table 104. LAI File Core Parameters

Parameter	Description
Pin Count	The **Pin Count** parameter signifies the number of pins you want dedicated to your LAI. The pins must be connected to a debug header on your board. Within the Intel-supported device, each pin is mapped to a user-configurable number of internal signals. The **Pin Count** parameter can range from 1 to 255 pins.
Bank Count	The **Bank Count** parameter signifies the number of internal signals that you want to map to each pin. For example, a **Bank Count** of 8 implies that you will connect eight internal signals to each pin. The **Bank Count** parameter can range from 1 to 255 banks.
Output/Capture Mode	The **Output/Capture Mode** parameter signifies the type of acquisition you perform. There are two options that you can select: **Combinational/Timing**—This acquisition uses your external logic analyzer's internal clock to determine when to sample data. Because **Combinational/Timing** acquisition samples data asynchronously to your Intel-supported device, you must determine the sample frequency you should use to debug and verify your system. This mode is effective if you want to measure timing information, such as channel-to-channel skew. For more information about the sampling frequency and the speeds at which it can run, refer to the data sheet for your external logic analyzer. **Registered/State**—This acquisition uses a signal from your system under test to determine when to sample. Because **Registered/State** acquisition samples data synchronously with your Intel-supported device, it provides you with a functional view of your Intel-supported device while it is running. This mode is effective when you verify the functionality of your design.
Clock	The **Clock** parameter is available only when **Output/Capture Mode** is set to **Registered State**. You must specify the sample clock in the **Core Parameters** view. The sample clock can be any signal in your design. However, for best results, Intel FPGA recommends that you use a clock with an operating frequency fast enough to sample the data you would like to acquire.
Power-Up State	The **Power-Up State** parameter specifies the power-up state of the pins you have designated for use with the LAI. You have the option of selecting tri-stated for all pins, or selecting a particular bank that you have enabled.

11.4.2 Mapping the LAI File Pins to Available I/O Pins

To configure the `.lai` file I/O pin parameters, select **Pins** in the **Setup View** list. To assign pin locations for the LAI, double-click the **Location** column next to the reserved pins in the **Name** column, and select a pin from the list. Once a pin is selected, you can right-click and locate in the Pin Planner.

Figure 205. Mapping LAI file Pins



Related Links

Managing Device I/O Pins
In *Quartus Prime Pro Edition Volume 2*

11.4.3 Mapping Internal Signals to the LAI Banks

After you have specified the number of banks to use in the **Core Parameters** settings page, you must assign internal signals for each bank in the LAI. Click the **Setup View** arrow and select **Bank n** or **All Banks**.

To view all of your bank connections, click **Setup View** and select **All Banks**.

11.4.4 Using the Node Finder

Before making bank assignments, right click the Node list and select **Add Nodes** to open the **Node Finder**. Find the signals that you want to acquire, then drag and drop the signals from the **Node Finder** dialog box into the bank **Setup View**. When adding signals, use **Signal Tap: pre-synthesis** for non-incrementally routed instances and **Signal Tap: post-fitting** for incrementally routed instances.

As you continue to make assignments in the bank **Setup View**, the schematic of your LAI in the **Logical View** pane begins to reflect your changes. Continue making assignments for each bank in the **Setup View** until you have added all of the internal signals for which you wish to acquire data.

Related Links

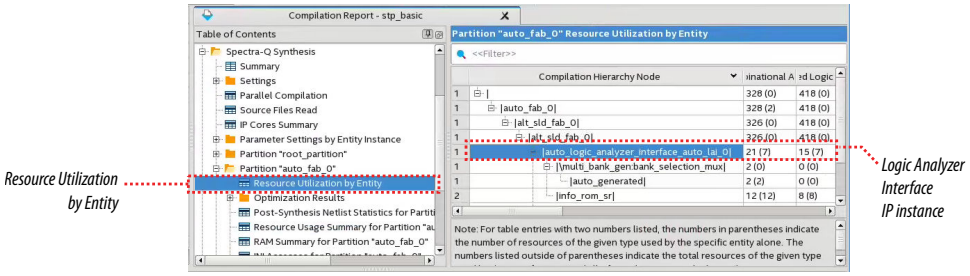
Node Finder Command
In Quartus Prime Help

11.4.5 Compiling Your Quartus Prime Project

After you save your `.lai` file, a dialog box prompts you to enable the Logic Analyzer Interface instance for the active project. Alternatively, you can define the `.lai` file your project uses in the **Global Project Settings** dialog box. After specifying the name of your `.lai` file, compile your project.

To verify the Logic Analyzer Interface is properly compiled with your project, open the **Compilation Report** tab and select Resource Utilization by Entity, nested under Partition "auto_fab_0". The LAI IP instance appears in the Compilation Hierarchy Node column, nested under the internal module of `auto_fab_0`

Figure 206. LAI Instance in Compilation Report



11.4.6 Programming Your Intel-Supported Device Using the LAI

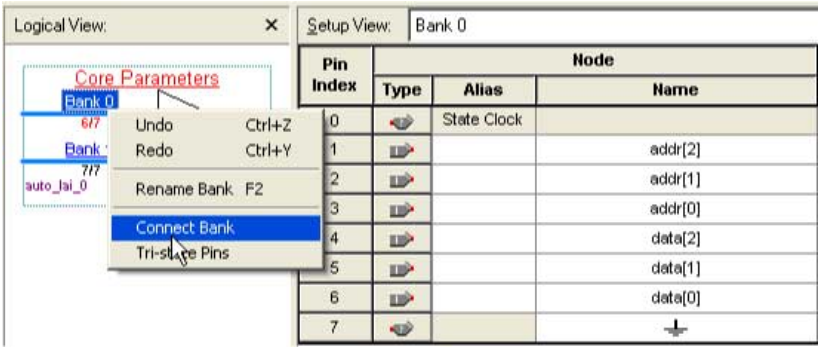
After compilation completes, you must configure your Intel-supported device before using the LAI.

You can use the LAI with multiple devices in your JTAG chain. Your JTAG chain can also consist of devices that do not support the LAI or non-Intel, JTAG-compliant devices. To use the LAI in more than one Intel-supported device, create an `.lai` file and configure an `.lai` file for each Intel-supported device that you want to analyze.

11.5 Controlling the Active Bank During Runtime

When you have programmed your Intel-supported device, you can control which bank you map to the reserved `.lai` file output pins. To control which bank you map, in the schematic in the Logical View, right-click the bank and click **Connect Bank**.

Figure 207. Configuring Banks



11.5.1 Acquiring Data on Your Logic Analyzer

To acquire data on your logic analyzer, you must establish a connection between your device and the external logic analyzer. For more information about this process and for guidelines about how to establish connections between debugging headers and logic analyzers, refer to the documentation for your logic analyzer.

11.6 Document Revision History

Table 105. Document Revision History

Date	Version	Changes
2017.05.08	17.0.0	• Updated *Compiling Your Quartus Prime Project* • Updated figure: LAI Instance in Compilation Report.
2016.10.31	16.1.0	• Implemented Intel rebranding.
2015.11.02	15.1.0	Changed instances of *Quartus II* to *Quartus Prime*.
June 2014	14.0.0	• Dita conversion • Added limitation about HPS I/O support
June 2012	12.0.0	Removed survey link
November 2011	10.1.1	Changed to new document template
December 2010	10.1.0	• Minor editorial updates • Changed to new document template
August 2010	10.0.1	Corrected links
July 2010	10.0.0	• Created links to the Quartus Prime Help • Editorial updates • Removed Referenced Documents section
November 2009	9.1.0	• Removed references to APEX devices • Editorial updates
March 2009	9.0.0	• Minor editorial updates • Removed Figures 15–4, 15–5, and 15–11 from 8.1 version
November 2008	8.1.0	Changed to 8-1/2 x 11 page size. No change to content
May 2008	8.0.0	• Updated device support list on page 15–3 • Added links to referenced documents throughout the chapter • Added "Referenced Documents" • Added reference to *Section V. In-System Debugging* • Minor editorial updates

Related Links

Altera Documentation Archive

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

12 In-System Modification of Memory and Constants

12.1 About the In-System Memory Content Editor

The Quartus Prime In-System Memory Content Editor allows to view and update memories and constants with the JTAG port connection at runtime.

The In-System Memory Content Editor provides access to dense and complex FPGA designs through the JTAG interface. You can then identify, test, and resolve issues with your design by testing changes to memory contents in the FPGA while your design is running.

When you use the In-System Memory Content Editor in conjunction with the Signal Tap Logic Analyzer, you can view and debug your design in the hardware lab.

The ability to read data from memories and constants allows you to identify the source of problems. The write capability allows you to bypass functional issues by writing expected data. For example, if a parity bit in your memory is incorrect, you can use the In-System Memory Content Editor to write the correct parity bit values into your RAM, allowing your system to continue functioning. To check the error handling functionality of your design, you can intentionally write incorrect parity bit values into your RAM.

Related Links

- System Debugging Tools Overview on page 146
- Design Debugging with the Signal Tap Logic Analyzer on page 258
- Megafunctions/LPM
  List of the types of memories and constants currently supported by the Quartus Prime software

12.2 Design Flow Using the In-System Memory Content Editor

To use the In-System Memory Content Editor, perform the following steps:

1. Identify the memories and constants that you want to access.
2. Edit the memories and constants to be run-time modifiable.
3. Perform a full compilation.
4. Program your device.
5. Launch the In-System Memory Content Editor.

12.3 Creating In-System Modifiable Memories and Constants

When you specify that a memory or constant is run-time modifiable, the Quartus Prime software changes the default implementation. A single-port RAM is converted to a dual-port RAM, and a constant is implemented in registers instead of look-up tables (LUTs). These changes enable run-time modification without changing the functionality of your design.

If you instantiate a memory or constant IP core directly with ports and parameters in VHDL or Verilog HDL, add or modify the `lpm_hint` parameter as follows:

In VHDL code, add the following:

```
lpm_hint => "ENABLE_RUNTIME_MOD = YES,
   INSTANCE_NAME = <instantiation name>";
```

In Verilog HDL code, add the following:

```
defparam <megafunction instance name>.lpm_hint =
   "ENABLE_RUNTIME_MOD = YES,
   INSTANCE_NAME = <instantiation name>";
```

12.4 Running the In-System Memory Content Editor

The In-System Memory Content Editor has three separate panes: the **Instance Manager**, the **JTAG Chain Configuration**, and the **Hex Editor**.

The **Instance Manager** pane displays all available run-time modifiable memories and constants in your FPGA device. The **JTAG Chain Configuration** pane allows you to program your FPGA and select the Intel FPGA device in the chain to update.

Using the In-System Memory Content Editor does not require that you open a project. The In-System Memory Content Editor retrieves all instances of run-time configurable memories and constants by scanning the JTAG chain and sending a query to the specific device selected in the **JTAG Chain Configuration** pane.

If you have more than one device with in-system configurable memories or constants in a JTAG chain, you can launch multiple In-System Memory Content Editors within the Quartus Prime software to access the memories and constants in each of the devices. Each In-System Memory Content Editor can access the in-system memories and constants in a single device.

12.4.1 Instance Manager

You can read and write to in-system memory with the **Instance Manager** pane. When you scan the JTAG chain to update the **Instance Manager** pane, you can view a list of all run-time modifiable memories and constants in the design. The **Instance Manager** pane displays the Index, Instance, Status, Width, Depth, Type, and Mode of each element in the list.

Note: In addition to the buttons available in the **Instance Manager** pane, you can read and write data by selecting commands from the **Processing** menu, or the right-click menu in the **Instance Manager** pane or **Hex Editor** pane.

The status of each instance is also displayed beside each entry in the **Instance Manager** pane. The status indicates if the instance is **Not running**, **Offloading data**, or **Updating data**. The health monitor provides information about the status of the editor.

The Quartus Prime software assigns a different index number to each in-system memory and constant to distinguish between multiple instances of the same memory or constant function. View the **In-System Memory Content Editor Settings** section of the Compilation Report to match an index number with the corresponding instance ID.

Related Links

Instance Manager Pane
In Quartus Prime Help

12.4.2 Editing Data Displayed in the Hex Editor Pane

You can edit data read from your in-system memories and constants displayed in the **Hex Editor** pane by typing values directly into the editor or by importing memory files.

12.4.3 Importing and Exporting Memory Files

The In-System Memory Content Editor allows you to import and export data values for memories that have the In-System Updating feature enabled. Importing from a data file enables you to quickly load an entire memory image. Exporting to a data file enables you to save the contents of the memory for future use.

12.4.4 Scripting Support

The In-System Memory Content Editor supports reading and writing of memory contents via a Tcl script or Tcl commands entered at a command prompt. For detailed information about scripting command options, refer to the Quartus Prime command-line and Tcl API Help browser.

To run the Help browser, type the following command at the command prompt:

```
quartus_sh --qhelp
```

The commonly used commands for the In-System Memory Content Editor are as follows:

- Reading from memory:

```
read_content_from_memory
[-content_in_hex]
-instance_index <instance index>
-start_address <starting address>
-word_count <word count>
```

- Writing to memory:

  ```
  write_content_to_memory
  ```

- Saving memory contents to a file:

  ```
  save_content_from_memory_to_file
  ```

- Updating memory contents from a file:

  ```
  update_content_to_memory_from_file
  ```

Related Links

- Tcl Scripting
  In *Quartus Prime Pro Edition Handbook Volume 2*
- Command Line Scripting
  In *Quartus Prime Pro Edition Handbook Volume 2*
- API Functions for Tcl
  In Quartus Prime Help

12.4.5 Programming the Device with the In-System Memory Content Editor

If you make changes to your design, you can program the device from within the In-System Memory Content Editor.

12.4.6 Example: Using the In-System Memory Content Editor with the Signal Tap Logic Analyzer

The following scenario describes how you can use the In-System Updating of Memory and Constants feature with the Signal Tap Logic Analyzer to efficiently debug your design. You can use the In-System Memory Content Editor and the Signal Tap Logic Analyzer simultaneously with the JTAG interface.

Scenario: After completing your FPGA design, you find that the characteristics of your FIR filter design are not as expected.

1. To locate the source of the problem, change all your FIR filter coefficients to be in-system modifiable and instantiate the Signal Tap Logic Analyzer.
2. Using the Signal Tap Logic Analyzer to tap and trigger on internal design nodes, you find the FIR filter to be functioning outside of the expected cutoff frequency.
3. Using the **In-System Memory Content Editor**, you check the correctness of the FIR filter coefficients. Upon reading each coefficient, you discover that one of the coefficients is incorrect.
4. Because your coefficients are in-system modifiable, you update the coefficients with the correct data with the **In-System Memory Content Editor**.

   In this scenario, you can quickly locate the source of the problem using both the In-System Memory Content Editor and the Signal Tap Logic Analyzer. You can also verify the functionality of your device by changing the coefficient values before modifying the design source files.

   You can also modify the coefficients with the In-System Memory Content Editor to vary the characteristics of the FIR filter, for example, filter attenuation, transition bandwidth, cut-off frequency, and windowing function.

12.5 Document Revision History

Table 106. Document Revision History

Date	Version	Changes
2016.10.31	16.1.0	• Implemented Intel rebranding.
2015.11.02	15.1.0	Changed instances of *Quartus II* to *Quartus Prime*.
June 2014	14.0.0	• Dita conversion. • Removed references to megafunction and replaced with IP core.
June 2012	12.0.0	Removed survey link.
November 2011	10.0.3	Template update.
December 2010	10.0.2	Changed to new document template. No change to content.
August 2010	10.0.1	Corrected links
July 2010	10.0.0	• Inserted links to Quartus Prime Help • Removed Reference Documents section
November 2009	9.1.0	• Delete references to APEX devices • Style changes
March 2009	9.0.0	No change to content
November 2008	8.1.0	Changed to 8-1/2 x 11 page size. No change to content.
May 2008	8.0.0	• Added reference to Section V. In-System Debugging in volume 3 of the Quartus Prime Handbook on page 16-1 • Removed references to the Mercury device, as it is now considered to be a "Mature" device • Added links to referenced documents throughout document • Minor editorial updates

Related Links

Altera Documentation Archive

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

13 Design Debugging Using In-System Sources and Probes

The Signal Tap Logic Analyzer and Signal Probe allow you to read or "tap" internal logic signals during run time as a way to debug your logic design.

Traditional debugging techniques often involve using an external pattern generator to exercise the logic and a logic analyzer to study the output waveforms during run time.

You can make the debugging cycle more efficient when you can drive any internal signal manually within your design, which allows you to perform the following actions:

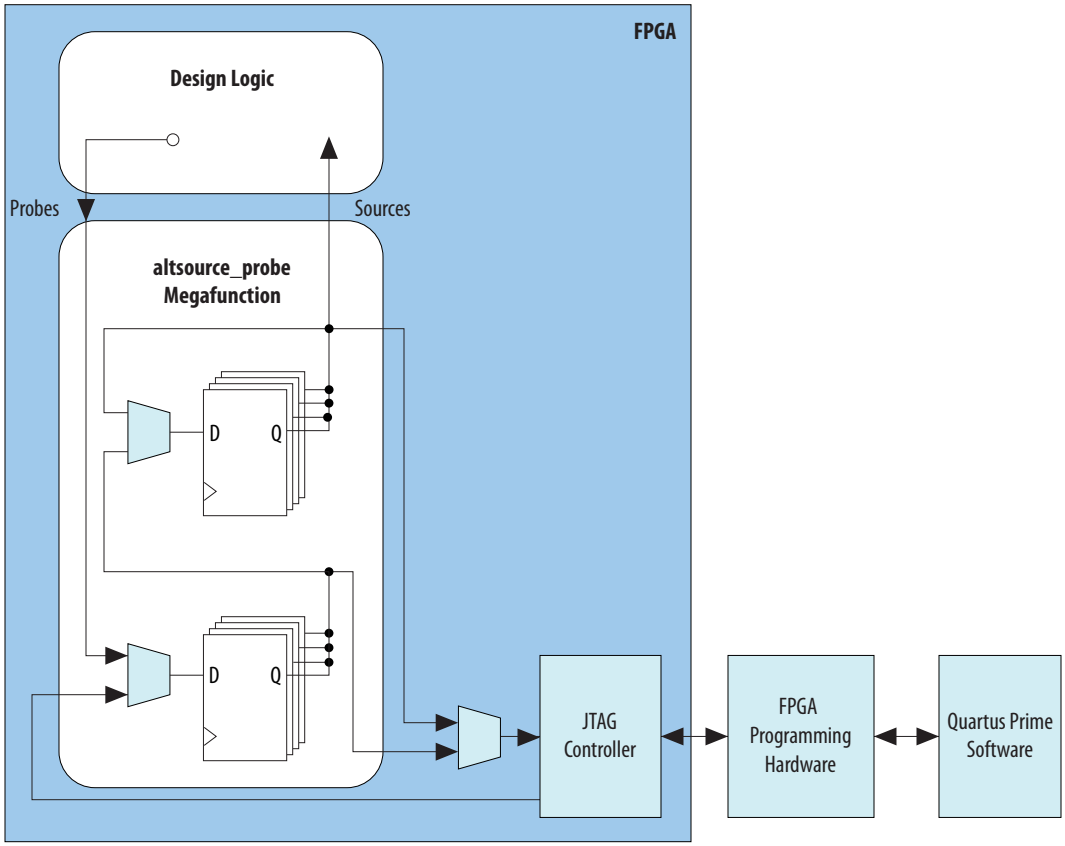
- Force the occurrence of trigger conditions set up in the Signal Tap Logic Analyzer
- Create simple test vectors to exercise your design without using external test equipment
- Dynamically control run time control signals with the JTAG chain

The In-System Sources and Probes Editor in the Quartus Prime software extends the portfolio of verification tools, and allows you to easily control any internal signal and provides you with a completely dynamic debugging environment. Coupled with either the Signal Tap Logic Analyzer or Signal Probe, the In-System Sources and Probes Editor gives you a powerful debugging environment in which to generate stimuli and solicit responses from your logic design.

The Virtual JTAG IP core and the In-System Memory Content Editor also give you the capability to drive virtual inputs into your design. The Quartus Prime software offers a variety of on-chip debugging tools.

The In-System Sources and Probes Editor consists of the ALTSOURCE_PROBE IP core and an interface to control the ALTSOURCE_PROBE IP core instances during run time. Each ALTSOURCE_PROBE IP core instance provides you with source output ports and probe input ports, where source ports drive selected signals and probe ports sample selected signals. When you compile your design, the ALTSOURCE_PROBE IP core sets up a register chain to either drive or sample the selected nodes in your logic design. During run time, the In-System Sources and Probes Editor uses a JTAG connection to shift data to and from the ALTSOURCE_PROBE IP core instances. The figure shows a block diagram of the components that make up the In-System Sources and Probes Editor.

Figure 208. In-System Sources and Probes Editor Block Diagram



The ALTSOURCE_PROBE IP core hides the detailed transactions between the JTAG controller and the registers instrumented in your design to give you a basic building block for stimulating and probing your design. Additionally, the In-System Sources and Probes Editor provides single-cycle samples and single-cycle writes to selected logic nodes. You can use this feature to input simple virtual stimuli and to capture the current value on instrumented nodes. Because the In-System Sources and Probes Editor gives you access to logic nodes in your design, you can toggle the inputs of low-level components during the debugging process. If used in conjunction with the Signal Tap Logic Analyzer, you can force trigger conditions to help isolate your problem and shorten your debugging process.

The In-System Sources and Probes Editor allows you to easily implement control signals in your design as virtual stimuli. This feature can be especially helpful for prototyping your design, such as in the following operations:

- Creating virtual push buttons
- Creating a virtual front panel to interface with your design
- Emulating external sensor data
- Monitoring and changing run time constants on the fly

The In-System Sources and Probes Editor supports Tcl commands that interface with all your ALTSOURCE_PROBE IP core instances to increase the level of automation.

Related Links

System Debugging Tools
For an overview and comparison of all the tools available in the Quartus Prime software on-chip debugging tool suite

13.1 Hardware and Software Requirements

The following components are required to use the In-System Sources and Probes Editor:

- Quartus Prime software

or

- Quartus Prime Lite Edition
- Download Cable (USB-Blaster™ download cable or ByteBlaster™ cable)
- Intel FPGA development kit or user design board with a JTAG connection to device under test

The In-System Sources and Probes Editor supports the following device families:

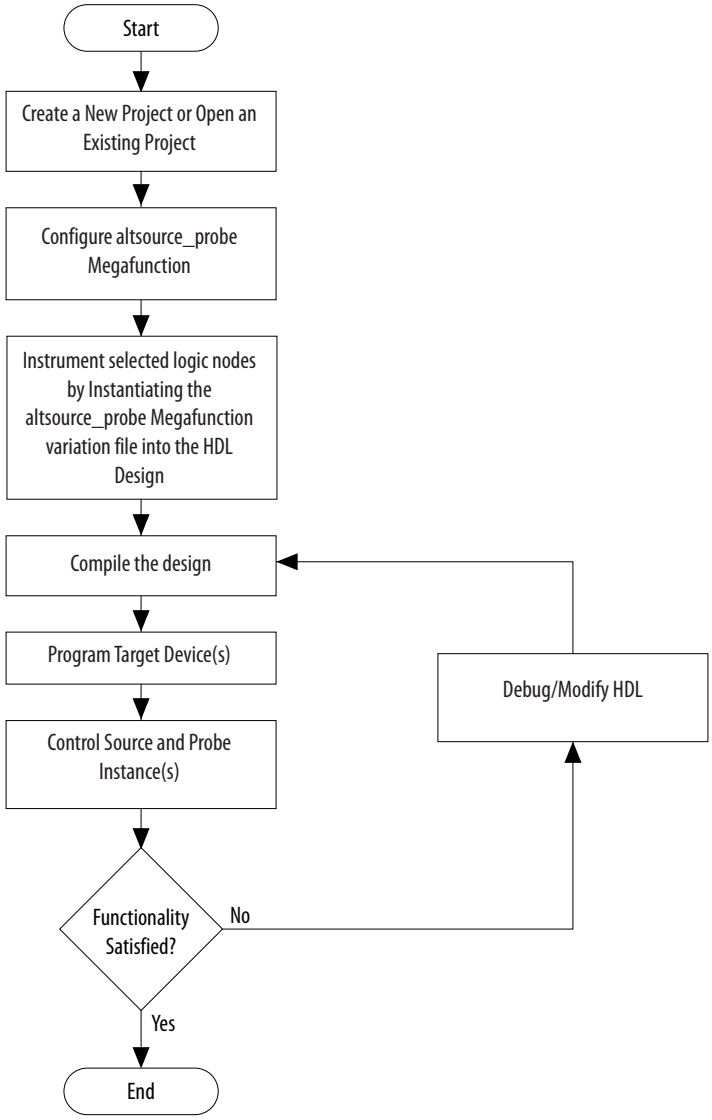
- Arria® series
- Stratix® series
- Cyclone® series
- MAX® series

13.2 Design Flow Using the In-System Sources and Probes Editor

The In-System Sources and Probes Editor supports an RTL flow. Signals that you want to view in the In-System Sources and Probes editor are connected to an instance of the In-System Sources and Probes IP core.

After you compile the design, you can control each instance via the **In-System Sources and Probes Editor** pane or via a Tcl interface.

Figure 209. FPGA Design Flow Using the In-System Sources and Probes Editor



13.2.1 Instantiating the In-System Sources and Probes IP Core

You must instantiate the In-System Sources and Probes IP core before you can use the In-System Sources and Probes editor. Use the IP Catalog and parameter editor to instantiate a custom variation of the In-System Sources and Probes IP core.

To configure the In-System Sources and Probes IP core, perform the following steps::

1. On the Tools menu, click **Tools > IP Catalog**.
2. Locate and double-click the In-System Sources and Probes IP core. The parameter editor appears.
3. Specify a name for your custom IP variation.
4. Specify the desired parameters for your custom IP variation. You can specify up to up to 512 bits for each source. Your design may include up to 128 instances of this IP core.
5. Click **Generate** or **Finish** to generate IP core synthesis and simulation files matching your specifications. The parameter editor generates the necessary variation files and the instantiation template based on your specification. Use the generated template to instantiate the In-System Sources and Probes IP core in your design.

   Note: The In-System Sources and Probes Editor does not support simulation. You must remove the In-System Sources and Probes IP core before you create a simulation netlist.

13.2.2 In-System Sources and Probes IP Core Parameters

Use the template to instantiate the variation file in your design.

Table 107. In-System Sources and Probes IP Port Information

Port Name	Required?	Direction	Comments
`probe[]`	No	Input	The outputs from your design.
`source_clk`	No	Input	Source Data is written synchronously to this clock. This input is required if you turn on **Source Clock** in the **Advanced Options** box in the parameter editor.
`source_ena`	No	Input	Clock enable signal for `source_clk`. This input is required if specified in the **Advanced Options** box in the parameter editor.
`source[]`	No	Output	Used to drive inputs to user design.

You can include up to 128 instances of the in-system sources and probes IP core in your design, if your device has available resources. Each instance of the IP core uses a pair of registers per signal for the width of the widest port in the IP core. Additionally, there is some fixed overhead logic to accommodate communication between the IP core instances and the JTAG controller. You can also specify an additional pair of registers per source port for synchronization.

When you compile your design that includes the In-System Sources and Probes IP core, the In-System Sources and Probes and SLD Hub Controller IP core are added to your compilation hierarchy automatically. These IP cores provide communication between the JTAG controller and your instrumented logic.

You can modify the number of connections to your design by editing the In-System Sources and Probes IP core. To open the design instance you want to modify in the parameter editor, double-click the instance in the Project Navigator. You can then modify the connections in the HDL source file. You must recompile your design after you make changes.

13.3 Compiling the Design

When you compile your design that includes the In-System Sources and ProbesIP core, the In-System Sources and Probes and SLD Hub Controller IP core are added to your compilation hierarchy automatically. These IP cores provide communication between the JTAG controller and your instrumented logic.

You can modify the number of connections to your design by editing the In-System Sources and Probes IP core. To open the design instance you want to modify in the parameter editor, double-click the instance in the Project Navigator. You can then modify the connections in the HDL source file. You must recompile your design after you make changes.

13.4 Running the In-System Sources and Probes Editor

The In-System Sources and Probes Editor gives you control over all ALTSOURCE_PROBE IP core instances within your design. The editor allows you to view all available run time controllable instances of the ALTSOURCE_PROBE IP core in your design, provides a push-button interface to drive all your source nodes, and provides a logging feature to store your probe and source data.

To run the In-System Sources and Probes Editor:

- On the **Tools** menu, click **In-System Sources and Probes Editor.**

13.4.1 In-System Sources and Probes Editor GUI

The In-System Sources and Probes Editor contains three panes:

- **JTAG Chain Configuration**—Allows you to specify programming hardware, device, and file settings that the In-System Sources and Probes Editor uses to program and acquire data from a device.
- **Instance Manager**—Displays information about the instances generated when you compile a design, and allows you to control data that the In-System Sources and Probes Editor acquires.
- **In-System Sources and Probes Editor**—Logs all data read from the selected instance and allows you to modify source data that is written to your device.

When you use the In-System Sources and Probes Editor, you do not need to open a Quartus Prime software project. The In-System Sources and Probes Editor retrieves all instances of the ALTSOURCE_PROBE IP core by scanning the JTAG chain and sending a query to the device selected in the **JTAG Chain Configuration** pane. You can also use a previously saved configuration to run the In-System Sources and Probes Editor.

Each **In-System Sources and Probes Editor** pane can access the ALTSOURCE_PROBE IP core instances in a single device. If you have more than one device containing IP core instances in a JTAG chain, you can launch multiple **In-System Sources and Probes Editor** panes to access the IP core instances in each device.

13.4.2 Programming Your Device With JTAG Chain Configuration

After you compile your project, you must configure your FPGA before you use the In-System Sources and Probes Editor.

To configure a device to use with the In-System Sources and Probes Editor, perform the following steps:

1. Open the In-System Sources and Probes Editor.
2. In the **JTAG Chain Configuration** pane, point to **Hardware,** and then select the hardware communications device. You may be prompted to configure your hardware; in this case, click **Setup**.
3. From the **Device** list, select the FPGA device to which you want to download the design (the device may be automatically detected). You may need to click **Scan Chain** to detect your target device.
4. In the **JTAG Chain Configuration** pane, click to browse for the SRAM Object File (**.sof**) that includes the In-System Sources and Probes instance or instances. (The **.sof** may be automatically detected).
5. Click **Program Device** to program the target device.

13.4.3 Instance Manager

The **Instance Manager** pane provides a list of all ALTSOURCE_PROBE instances in the design and allows you to configure how data is acquired from or written to those instances.

The following buttons and sub-panes are provided in the **Instance Manager** pane:

- **Read Probe Data**—Samples the probe data in the selected instance and displays the probe data in the **In-System Sources and Probes Editor** pane.
- **Continuously Read Probe Data**—Continuously samples the probe data of the selected instance and displays the probe data in the **In-System Sources and Probes Editor** pane; you can modify the sample rate via the **Probe read interval** setting.
- **Stop Continuously Reading Probe Data**—Cancels continuous sampling of the probe of the selected instance.
- **Write Source Data**—Writes data to all source nodes of the selected instance.
- **Probe Read Interval**—Displays the sample interval of all the In-System Sources and Probe instances in your design; you can modify the sample interval by clicking **Manual.**
- **Event Log**—Controls the event log in the **In-System Sources and Probes Editor** pane.
- **Write Source Data**—Allows you to manually or continuously write data to the system.

The status of each instance is also displayed beside each entry in the **Instance Manager** pane. The status indicates if the instance is **Not running Offloading data**, **Updating data**, or if an **Unexpected JTAG communication error** occurs. This status indicator provides information about the sources and probes instances in your design.

13.4.4 In-System Sources and Probes Editor Pane

The **In-System Sources and Probes Editor** pane allows you to view data from all sources and probes in your design.

The data is organized according to the index number of the instance. The editor provides an easy way to manage your signals, and allows you to rename signals or group them into buses. All data collected from in-system source and probe nodes is recorded in the event log and you can view the data as a timing diagram.

13.4.4.1 Reading Probe Data

You can read data by selecting the ALTSOURCE_PROBE instance in the **Instance Manager** pane and clicking **Read Probe Data**.

This action produces a single sample of the probe data and updates the data column of the selected index in the **In-System Sources and Probes Editor** pane. You can save the data to an event log by turning on the **Save data to event log** option in the **Instance Manager** pane.

If you want to sample data from your probe instance continuously, in the **Instance Manager** pane, click the instance you want to read, and then click **Continuously read probe data**. While reading, the status of the active instance shows **Unloading**. You can read continuously from multiple instances.

You can access read data with the shortcut menus in the **Instance Manager** pane.

To adjust the probe read interval, in the **Instance Manager** pane, turn on the **Manual** option in the **Probe read interval** sub-pane, and specify the sample rate in the text field next to the **Manual** option. The maximum sample rate depends on your computer setup. The actual sample rate is shown in the **Current interval** box. You can adjust the event log window buffer size in the **Maximum Size** box.

13.4.4.2 Writing Data

To modify the source data you want to write into the ALTSOURCE_PROBE instance, click the name field of the signal you want to change. For buses of signals, you can double-click the data field and type the value you want to drive out to the ALTSOURCE_PROBE instance. The In-System Sources and Probes Editor stores the modified source data values in a temporary buffer.

Modified values that are not written out to the ALTSOURCE_PROBE instances appear in red. To update the ALTSOURCE_PROBE instance, highlight the instance in the **Instance Manager** pane and click **Write source data**. The **Write source data** function is also available via the shortcut menus in the **Instance Manager** pane.

The In-System Sources and Probes Editor provides the option to continuously update each ALTSOURCE_PROBE instance. Continuous updating allows any modifications you make to the source data buffer to also write immediately to the ALTSOURCE_PROBE instances. To continuously update the ALTSOURCE_PROBE instances, change the **Write source data** field from **Manually** to **Continuously**.

13.4.4.3 Organizing Data

The **In-System Sources and Probes Editor** pane allows you to group signals into buses, and also allows you to modify the display options of the data buffer.

To create a group of signals, select the node names you want to group, right-click and select **Group**. You can modify the display format in the Bus Display Format and the Bus Bit order shortcut menus.

The **In-System Sources and Probes Editor** pane allows you to rename any signal. To rename a signal, double-click the name of the signal and type the new name.

The event log contains a record of the most recent samples. The buffer size is adjustable up to 128k samples. The time stamp for each sample is logged and is displayed above the event log of the active instance as you move your pointer over the data samples.

You can save the changes that you make and the recorded data to a Sources and Probes File (**.spf**). To save changes, on the File menu, click **Save**. The file contains all the modifications you made to the signal groups, as well as the current data event log.

13.5 Tcl interface for the In-System Sources and Probes Editor

To support automation, the In-System Sources and Probes Editor supports the procedures described in this chapter in the form of Tcl commands. The Tcl package for the In-System Sources and Probes Editor is included by default when you run **quartus_stp**.

The Tcl interface for the In-System Sources and Probes Editor provides a powerful platform to help you debug your design. The Tcl interface is especially helpful for debugging designs that require toggling multiple sets of control inputs. You can combine multiple commands with a Tcl script to define a custom command set.

Table 108. In-System Sources and Probes Tcl Commands

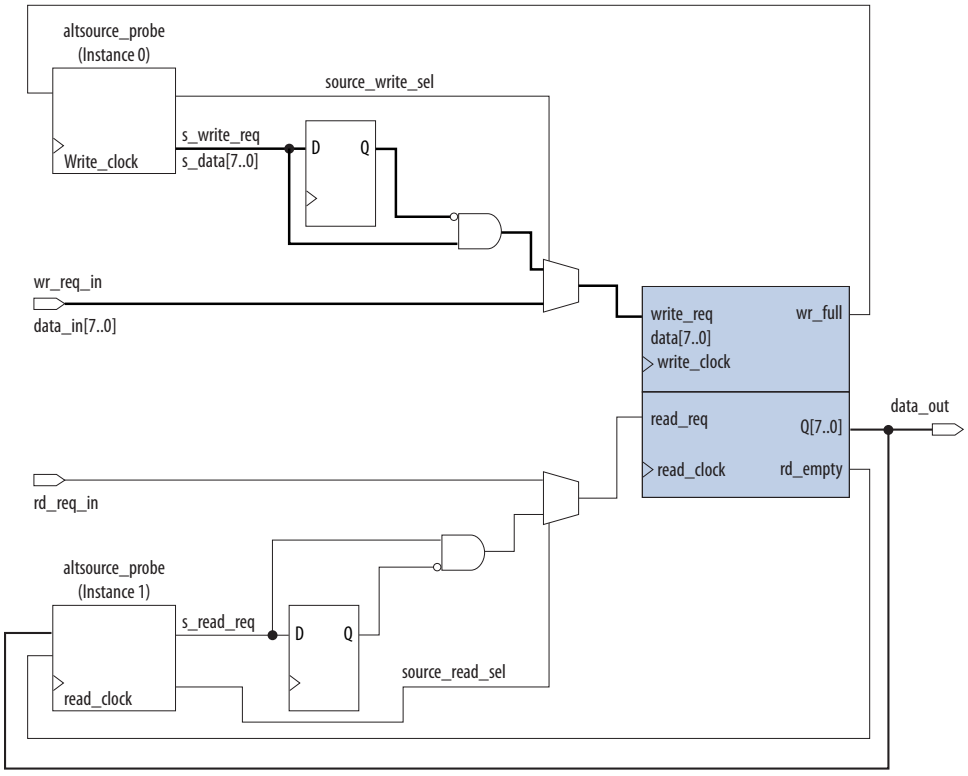
Command	Argument	Description
`start_insystem_source_probe`	`-device_name` *<device name>* `-hardware_name` *<hardware name>*	Opens a handle to a device with the specified hardware. Call this command before starting any transactions.
`get_insystem_source_probe_instance_info`	`-device_name` *<device name>* -hardware_name *<hardware name>*	Returns a list of all ALTSOURCE_PROBE instances in your design. Each record returned is in the following format: {*<instance Index>*, *<source width>*, *<probe width>*, *<instance name>*}
`read_probe_data`	`-instance_index` *<instance_index>* `-value_in_hex` (optional)	Retrieves the current value of the probe. A string is returned that specifies the status of each probe, with the MSB as the left-most bit.
`read_source_data`	`-instance_index` *<instance_index>* `-value_in_hex` (optional)	Retrieves the current value of the sources. A string is returned that specifies the status of each source, with the MSB as the left-most bit.
`write_source_data`	`-instance_index` *<instance_index>* `-value` *<value>* `-value_in_hex` (optional)	Sets the value of the sources. A binary string is sent to the source ports, with the MSB as the left-most bit.
`end_interactive_probe`	None	Releases the JTAG chain. Issue this command when all transactions are finished.

The example shows an excerpt from a Tcl script with procedures that control the ALTSOURCE_PROBE instances of the design as shown in the figure below. The example design contains a DCFIFO with ALTSOURCE_PROBE instances to read from

and write to the DCFIFO. A set of control muxes are added to the design to control the flow of data to the DCFIFO between the input pins and the ALTSOURCE_PROBE instances. A pulse generator is added to the read request and write request control lines to guarantee a single sample read or write. The ALTSOURCE_PROBE instances, when used with the script in the example below, provide visibility into the contents of the FIFO by performing single sample write and read operations and reporting the state of the full and empty status flags.

Use the Tcl script in debugging situations to either empty or preload the FIFO in your design. For example, you can use this feature to preload the FIFO to match a trigger condition you have set up within the Signal Tap Logic Analyzer.

Figure 210. DCFIFO Example Design Controlled by Tcl Script




```
## Setup USB hardware  - assumes only USB Blaster is installed and
## an FPGA is the only device in the JTAG chain
set usb [lindex [get_hardware_names] 0]
set device_name [lindex [get_device_names -hardware_name $usb] 0]
## write procedure :  argument value is integer
proc write {value} {
global device_name usb
variable full
start_insystem_source_probe -device_name $device_name -hardware_name $usb
#read full flag
set full [read_probe_data -instance_index 0]
if {$full == 1} {end_insystem_source_probe
return "Write Buffer Full"
}
##toggle select line, drive value onto port, toggle enable
##bits 7:0 of instance 0 is S_data[7:0]; bit 8 = S_write_req;
##bit 9 = Source_write_sel
##int2bits is custom procedure that returns a bitstring from an integer
```

```
    ## argument
write_source_data -instance_index 0 -value /[int2bits [expr 0x200 | $value]]
write_source_data -instance_index 0 -value [int2bits [expr 0x300 | $value]]
##clear transaction
write_source_data -instance_index 0 -value 0
end_insystem_source_probe
}
proc read {} {
global device_name usb
variable empty
start_insystem_source_probe -device_name $device_name -hardware_name $usb
##read empty flag : probe port[7:0] reads FIFO output; bit 8 reads empty_flag
set empty [read_probe_data -instance_index 1]
if {[regexp {1........} $empty]} { end_insystem_source_probe
return "FIFO empty" }
## toggle select line for read transaction
## Source_read_sel = bit 0; s_read_reg = bit 1
## pulse read enable on DC FIFO
write_source_data -instance_index 1 -value 0x1 -value_in_hex
write_source_data -instance_index 1 -value 0x3 -value_in_hex
set x [read_probe_data -instance_index 1 ]
end_insystem_source_probe
return $x
}
```

Related Links

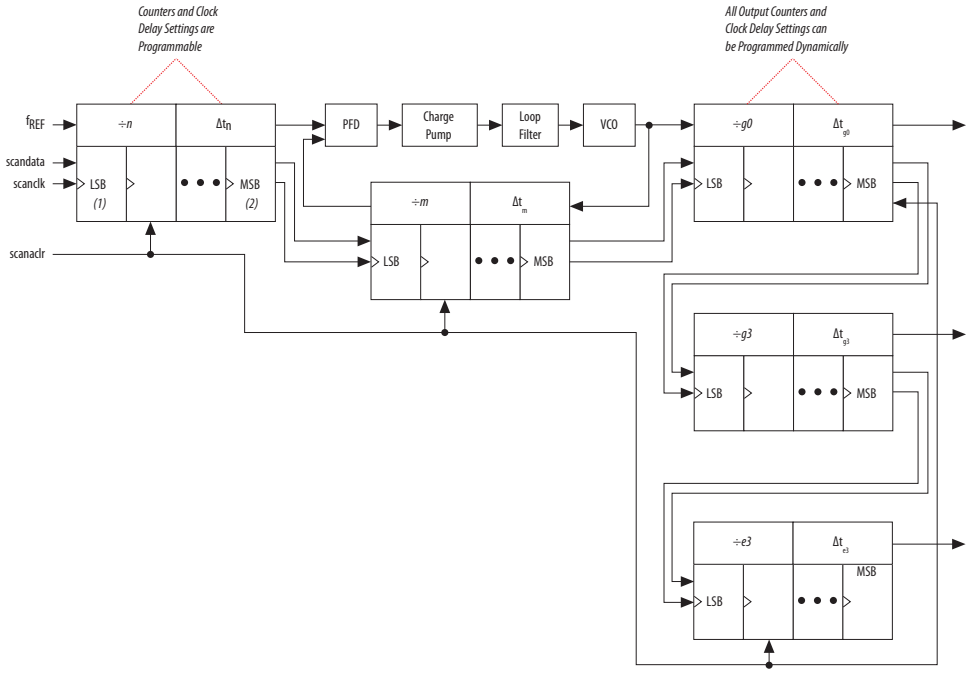
- Tcl Scripting
- Quartus Prime Settings File Manual
- Command Line Scripting

13.6 Design Example: Dynamic PLL Reconfiguration

The In-System Sources and Probes Editor can help you create a virtual front panel during the prototyping phase of your design. You can create relatively simple, high functioning designs of in a short amount of time. The following PLL reconfiguration example demonstrates how to use the In-System Sources and Probes Editor to provide a GUI to dynamically reconfigure a Stratix PLL.

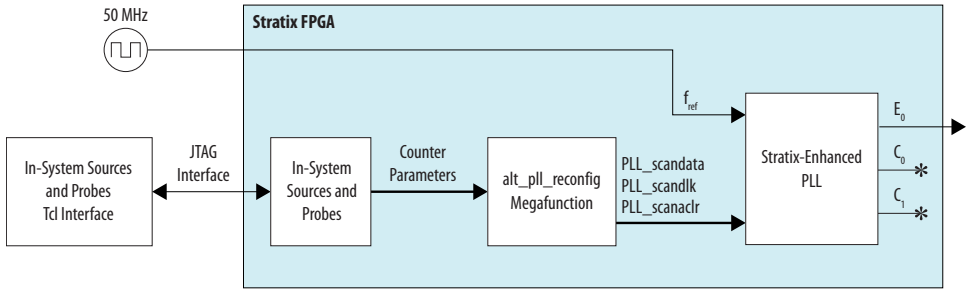
Stratix PLLs allow you to dynamically update PLL coefficients during run time. Each enhanced PLL within the Stratix device contains a register chain that allows you to modify the pre-scale counters (m and n values), output divide counters, and delay counters. In addition, the ALTPLL_RECONFIG IP core provides an easy interface to access the register chain counters. The ALTPLL_RECONFIG IP core provides a cache that contains all modifiable PLL parameters. After you update all the PLL parameters in the cache, the ALTPLL_RECONFIG IP core drives the PLL register chain to update the PLL with the updated parameters. The figure shows a Stratix-enhanced PLL with reconfigurable coefficients.

Figure 211. Stratix-Enhanced PLL with Reconfigurable Coefficients



The following design example uses an ALTSOURCE_PROBE instance to update the PLL parameters in the ALTPLL_RECONFIG IP core cache. The ALTPLL_RECONFIG IP core connects to an enhanced PLL in a Stratix FPGA to drive the register chain containing the PLL reconfigurable coefficients. This design example uses a Tcl/Tk script to generate a GUI where you can enter in new m and n values for the enhanced PLL. The Tcl script extracts the m and n values from the GUI, shifts the values out to the ALTSOURCE_PROBE instances to update the values in the ALTPLL_RECONFIG IP core cache, and asserts the reconfiguration signal on the ALTPLL_RECONFIG IP core. The reconfiguration signal on the ALTPLL_RECONFIG IP core starts the register chain transaction to update all PLL reconfigurable coefficients.

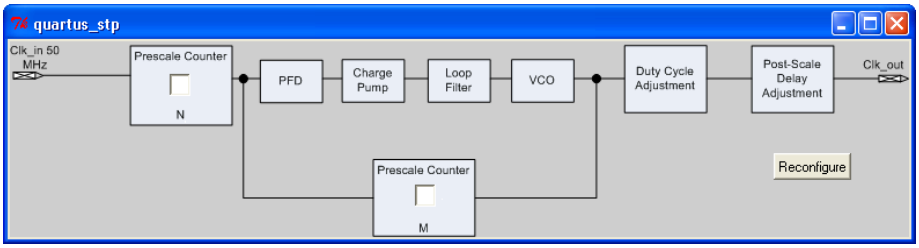
Figure 212. Block Diagram of Dynamic PLL Reconfiguration Design Example



This design example was created using a Nios II Development Kit, Stratix Edition. The file `sourceprobe_DE_dynamic_pll.zip` contains all the necessary files for running this design example, including the following:

- `Readme.txt`—A text file that describes the files contained in the design example and provides instructions about running the Tk GUI shown in the figure below.
- `Interactive_Reconfig.qar`—The archived Quartus Prime project for this design example.

Figure 213. Interactive PLL Reconfiguration GUI Created with Tk and In-System Sources and Probes Tcl Package



Related Links

On-chip Debugging Design Examples
to download the In-System Sources and Probes Example

13.7 Document Revision History

Table 109. Document Revision History

Date	Version	Changes
2016.10.31	16.1.0	• Implemented Intel rebranding.
2015.11.02	15.1.0	Changed instances of *Quartus II* to *Quartus Prime*.
June 2014	14.0.0	Updated formatting.
June 2012	12.0.0	Removed survey link.
November 2011	10.1.1	Template update.
December 2010	10.1.0	Minor corrections. Changed to new document template.
July 2010	10.0.0	Minor corrections.
November 2009	9.1.0	• Removed references to obsolete devices. • Style changes.
March 2009	9.0.0	No change to content.
November 2008	8.1.0	Changed to 8-1/2 x 11 page size. No change to content.
May 2008	8.0.0	• Documented that this feature does not support simulation on page 17–5 • Updated Figure 17–8 for Interactive PLL reconfiguration manager • Added hyperlinks to referenced documents throughout the chapter • Minor editorial updates

Related Links

Altera Documentation Archive

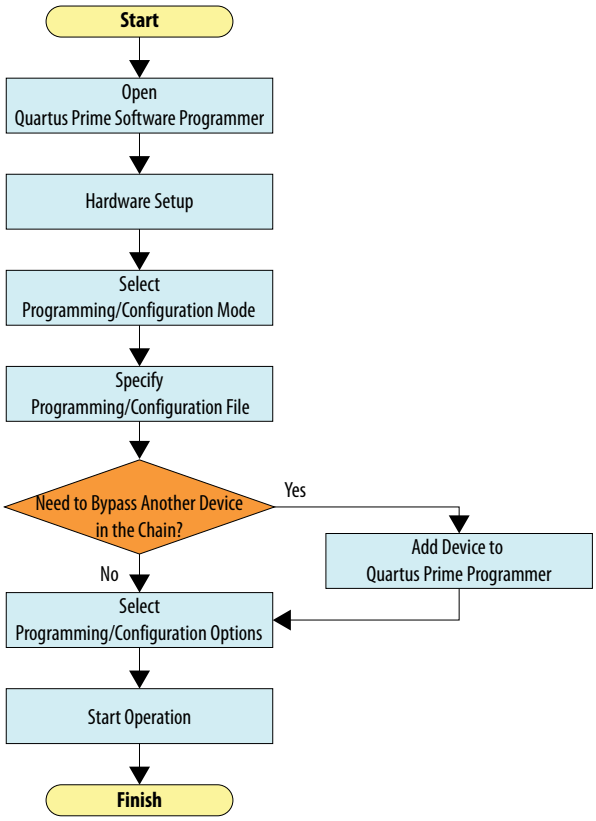
For previous versions of the *Quartus Prime Handbook*, search the Altera documentation archives.

14 Programming Intel FPGA Devices

The Quartus Prime Programmer allows you to program and configure Intel FPGA CPLD, FPGA, and configuration devices. After compiling your design, use the Quartus Prime Programmer to program or configure your device, to test the functionality of the design on a circuit board.

14.1 Programming Flow

Figure 214. Programming Flow



The following steps describe the programming flow:

1. Compile your design, such that the Quartus Prime Assembler generates the programming or configuration file.
2. Convert the programming or configuration file to target your configuration device and, optionally, create secondary programming files.

Table 110. Programming and Configuration File Format

File Format	FPGA	CPLD	Configuration Device	Serial Configuration Device
SRAM Object File (.sof)	Yes	—	—	—
Programmer Object File (.pof)	—	Yes	Yes	Yes
JEDEC JESD71 STAPL Format File (.jam)	Yes	Yes	Yes	—
Jam Byte Code File (.jbc)	Yes	Yes	Yes	—

3. Program and configure the FPGA, CPLD, or configuration device using the programming or configuration file with the Quartus Prime Programmer.

14.1.1 Stand-Alone Quartus Prime Programmer

Intel FPGA offers the free stand-alone Programmer, which has the same full functionality as the Quartus Prime Programmer in the Quartus Prime software. The stand-alone Programmer is useful when programming your devices with another workstation, so you do not need two full licenses. You can download the stand-alone Programmer from the Download Center on the Altera website.

Stand-Alone Programmer Memory Limitations

The stand-alone Programmer may use significant memory during the following operations:

- During auto-detect operations
- When the programming file is added to the flash
- During manual attachment of the flash into the Programmer window

The 32-bit stand-alone Programmer can only use a limited amount of memory when launched in 32-bit Windows. Note the following specific limitations of 32-bit stand-alone Programmer:

Table 111. Stand-Alone Programmer Memory Limitations

Application	Maximum Flash Device Size	Flash Device Operation Using PFL
32-bit Stand-Alone Programmer	Up to 512 Mb	Single Flash Device
64-bit Stand-Alone Programmer	Up to 2 Gb	Multiple Flash Device

The stand-alone Programmer supports combination and/or conversion of Quartus Prime programming files using the **Convert Programming Files** dialog box. You can convert programming files, such as Mask Settings File (.msf), Partial-Mask SRAM Object File (.pmsf), SRAM Object Files (.sof), or Programmer Object Files (.pof) into other file formats that support device configuration schemes for Intel FPGA devices.

Related Links

Download Center

14.1.2 Optional Programming or Configuration Files

The Quartus Prime software can generate optional programming or configuration files in various formats that you can use with programming tools other than the Quartus Prime Programmer. When you compile a design in the Quartus Prime software, the Assembler automatically generates either a `.sof` or `.pof`. The Assembler also allows you to convert FPGA configuration files to programming files for configuration devices.

Related Links

AN 425: Using Command-Line Jam STAPL Solution for Device Programming

14.1.3 Secondary Programming Files

The Quartus Prime software generates programming files in various formats for use with different programming tools.

Table 112. File Types Generated by the Quartus Prime Software and Supported by the Quartus Prime Programmer

File Type	Generated by the Quartus Prime Software	Supported by the Quartus Prime Programmer
`.sof`	Yes	Yes
`.pof`	Yes	Yes
`.jam`	Yes	Yes
`.jbc`	Yes	Yes
JTAG Indirect Configuration File (`.jic`)	Yes	Yes
Serial Vector Format File (`.svf`)	Yes	—
Hexadecimal (Intel-Format) Output File (`.hexout`)	Yes	—
Raw Binary File (`.rbf`)	Yes	Yes [8]
Tabular Text File (`.ttf`)	Yes	—
Raw Programming Data File (`.rpd`)	Yes	—

14.2 Quartus Prime Programmer GUI

The Quartus Prime **Programmer** window allows you to perform the following tasks:

- Add your programming and configuration files.
- Specify programming options and hardware.
- Start the programming or configuration of the device.

To open the **Programmer** window, click **Tools ➤ Programmer**. As you proceed through the programming flow, the Quartus Prime **Message** window reports the status of each operation.

8 Raw Binary File (`.rbf`) is supported by the Quartus Prime Programmer in Passive Serial (PS) configuration mode.

Related Links

Programmer Page (Options Dialog Box)
In Quartus Prime Help

14.2.1 Editing the Details of an Unknown Device

When the Quartus Prime Programmer automatically detects devices with shared JTAG IDs, the Programmer prompts you to specify the device in the JTAG chain. If the Programmer does not prompt you to specify the device, you must manually add each device in the JTAG chain to the Programmer, and define the instruction register length of each device.

To edit the details of an unknown device, follow these steps:

1. Double-click the unknown device listed under the device column.
2. Click **Edit**.
3. Change the device **Name**.
4. Specify the **Instruction register Length**.
5. Click **OK**.
6. Save the `.cdf` file.

14.2.2 Setting Up Your Hardware

Before you can program or configure your device, you must have the correct hardware setup. The Quartus Prime Programmer provides the flexibility to choose a download cable or programming hardware.

14.2.3 Setting the JTAG Hardware

The JTAG server allows the Quartus Prime Programmer to access the JTAG hardware. You can also access the JTAG download cable or programming hardware connected to a remote computer through the JTAG server of that computer. With the JTAG server, you can control the programming or configuration of devices from a single computer through other computers at remote locations. The JTAG server uses the TCP/IP communications protocol.

14.2.3.1 Running JTAG Daemon with Linux

The JTAGD daemon allows a remote machine to program or debug a board that is connected to a Linux host over the network. The JTAGD daemon also allows multiple programs to use JTAG resources at the same time. The JTAGD daemon is the Linux version of a JTAG server.

Run the JTAGD daemon to avoid:

- The JTAGD server from exiting after two minutes of idleness.
- The JTAGD server from not accepting connections from remote machines, which might lead to an intermittent failure.

To run JTAGD as a daemon, follow these steps:

1. Create an `/etc/jtagd` directory.
2. Set the permissions of this directory and the files in the directory to allow you to have the read/write access.
3. Run `jtagd` (with no arguments) from your `quartus/bin` directory.

The JTAGD daemon is now running and does not terminate when you log off.

14.2.4 Using the JTAG Chain Debugger Tool

The JTAG Chain Debugger tool allows you to test the JTAG chain integrity and detect intermittent failures of the JTAG chain. In addition, the tool allows you to shift in JTAG instructions and data through the JTAG interface, and step through the test access port (TAP) controller state machine for debugging purposes. You access the tool by clicking **Tools ➤ JTAG Chain Debugger** on the Quartus Prime software.

14.3 Programming and Configuration Modes

The following table lists the programming and configuration modes supported by Intel FPGA devices.

Table 113. Programming and Configuration Modes

Configuration Mode Supported by the Quartus Prime Programmer	FPGA	CPLD	Configuration Device	Serial Configuration Device
JTAG	Yes	Yes	Yes	—
Passive Serial (PS)	Yes	—	—	—
Active Serial (AS) Programming	—	—	—	Yes
Configuration via Protocol (CvP)	Yes	—	—	—
In-Socket Programming	—	Yes	Yes	Yes

Related Links

Configuration via Protocol (CvP) Implementation in V-series Intel FPGAs Devices User Guide

Describes the CvP configuration mode.

14.4 Design Security Keys

The Quartus Prime Programmer supports the generation of encryption key programming files and encrypted configuration files for Intel FPGAs that support the design security feature. You can also use the Quartus Prime Programmer to program the encryption key into the FPGA.

Related Links

AN 556: Using the Design Security Features in Intel FPGAs

14.5 Project Hash

The project hash feature allows you to verify if two programming files correspond to a compilation of the same set of source files. During compilation, the Quartus Prime software generates a unique project hash and embeds this value in programming files (`.sof`).

The project hash doesn't change for different builds of the Quartus Prime software, or when you install a software patch. However, if you upgrade any IP with a different build or patch, the project hash changes.

14.5.1 Obtaining Project Hash for Arria 10 Devices

To obtain the project hash value of a `.sof` programming file for a design targeted to Arria 10 devices, create a file named `project_hash.tcl`. In your file, copy and paste the following code:

```
############################################
## Begin project_hash.tcl
#
##
## @copyright Intel 2017
##
proc main_run {} {
global quartus
set qargs $quartus(args)
set nargs [llength $qargs]
load_package asm2
load_devices
set handle -1
set sof_file [lindex $qargs 0]
if [file exists $sof_file] {
set handle [open_sof $sof_file]
}
print "/metadata/0/project_hash"
if { $handle != -1 } {
close_handle $handle
}
}
############################################
main_run
## End of project_hash.tcl
############################################
```

Save the `project_hash.tcl` file in the same directory that contains your programming file, and type in the command line:

```
quartus_asm -t project_hash.tcl <sof-file>
```

The script prints the project hash value to the command line output.

Example 29. Output of Project Hash Extraction:

In this example, the programming file is `worm.sof`.

```
Info: ******************************************************************
Info: Running Quartus Prime Assembler
Info: Version 17.0.0 Build 288 04/12/2017 SJ Pro Edition
Info: Copyright (C) 2017  Intel Corporation. All rights reserved.
Info: Your use of Intel Corporation's design tools, logic functions
```

```
Info: and other software and tools, and its AMPP partner logic
Info: functions, and any output files from any of the foregoing
Info: (including device programming or simulation files), and any
Info: associated documentation or information are expressly subject
Info: to the terms and conditions of the Intel Program License
Info: Subscription Agreement, the Intel Quartus Prime License Agreement,
Info: the Intel MegaCore Function License Agreement, or other
Info: applicable license agreement, including, without limitation,
Info: that your use is for the sole purpose of programming logic
Info: devices manufactured by Intel and sold by Intel or its
Info: authorized distributors.  Please refer to the applicable
Info: agreement for further details.
Info: Processing started: Fri Apr 14 18:01:47 2017
Info: Command: quartus_asm -t project_hash.tcl worm.sof
Info: Quartus(args): worm.sof
0x1ffdc3f47c57bbe0075f6d4cb2cb9deb
Info (23030): Evaluation of Tcl script project_hash.tcl was successful
Info: Quartus Prime Assembler was successful. 0 errors, 0 warnings
Info: Peak virtual memory: 1451 megabytes
Info: Processing ended: Fri Apr 14 18:01:56 2017
Info: Elapsed time: 00:00:09
Info: Total CPU time (on all processors): 00:00:04
```

14.6 Convert Programming Files Dialog Box

The **Convert Programming Files** dialog box in the Programmer allows you to convert programming files from one file format to another. To access the **Convert Programming Files** dialog box, click **File ➤ Convert Programming Files...** on the Quartus Prime software.

For example, to store the FPGA data in configuration devices, you can convert the `.sof` data to another format, such as `.pof`, `.hexout`, `.rbf`, `.rpd`, or `.jic`, and then program the configuration device.

You can also configure multiple devices with an external host, such as a microprocessor or CPLD. For example, you can combine multiple `.sof` files into one `.pof` file. To save time in subsequent conversions, click **Save Conversion Setup** to save your conversion specifications in a Conversion Setup File (`.cof`). Click **Open Conversion Setup Data** to load your `.cof` setup in the **Convert Programming Files** dialog box.

Example 30. Conversion Setup File Contents

```
<?xml version="1.0" encoding="US-ASCII" standalone="yes"?>
<cof>
    <output_filename>output_file.pof</output_filename>
    <n_pages>1</n_pages>
    <width>1</width>
    <mode>14</mode>
    <sof_data>
        <user_name>Page_0</user_name>
        <page_flags>1</page_flags>
        <bit0>
            <sof_filename>/users/jbrossar/template/output_files/
template_test.sof</sof_filename>
        </bit0>
    </sof_data>
    <version>7</version>
    <create_cvp_file>0</create_cvp_file>
    <create_hps_iocsr>0</create_hps_iocsr>
    <auto_create_rpd>0</auto_create_rpd>
    <options>
        <map_file>1</map_file>
```

```
    </options>
    <MAX10_device_options>
        <por>0</por>
        <io_pullup>1</io_pullup>
        <auto_reconfigure>1</auto_reconfigure>
        <isp_source>0</isp_source>
        <verify_protect>0</verify_protect>
        <epof>0</epof>
        <ufm_source>0</ufm_source>
    </MAX10_device_options>
    <advanced_options>
        <ignore_epcs_id_check>0</ignore_epcs_id_check>
        <ignore_condone_check>2</ignore_condone_check>
        <plc_adjustment>0</plc_adjustment>
        <post_chain_bitstream_pad_bytes>-1</post_chain_bitstream_pad_bytes>
        <post_device_bitstream_pad_bytes>-1</post_device_bitstream_pad_bytes>
        <bitslice_pre_padding>1</bitslice_pre_padding>
    </advanced_options>
</cof>
```

Related Links

Convert Programming Files Dialog Box
In Quartus Prime Help

14.6.1 Debugging Your Configuration

Use the **Advanced** option in the **Convert Programming Files** dialog box to debug your configuration. You must choose the advanced settings that apply to your Intel FPGA device. You can direct the Quartus Prime software to enable or disable an advanced option by turning the option on or off in the **Advanced Options** dialog box. When you change settings in the **Advanced Options** dialog box, the change affects `.pof`, `.jic`, `.rpd`, and `.rbf` files.

The following table lists the **Advanced Options** settings in more detail:

Table 114. Advanced Options Settings

Option Setting	Description
Disable EPCS/EPCQ ID check	FPGA skips the EPCS/EPCQ silicon ID verification. Default setting is unavailable (EPCS/EPCQ ID check is enabled). Applies to the single- and multi-device AS configuration modes on all FPGA devices.
Disable AS mode CONF_DONE error check	FPGA skips the CONF_DONE error check. Default setting is unavailable (AS mode CONF_DONE error check is enabled). Applies to single- and multi-device (AS) configuration modes on all FPGA devices.
Program Length Count adjustment	Specifies the offset you can apply to the computed PLC of the entire bitstream. Default setting is 0. The value must be an integer. Applies to single- and multi-device (AS) configuration modes on all FPGA devices.
Post-chain bitstream pad bytes	Specifies the number of pad bytes appended to the end of an entire bitstream.

continued...

Option Setting	Description
	Default value is set to 0 if the bitstream of the last device is uncompressed. Set to 2 if the bitstream of the last device is compressed.
Post-device bitstream pad bytes	Specifies the number of pad bytes appended to the end of the bitstream of a device. Default value is 0. No negative integer. Applies to all single-device configuration modes on all FPGA devices.
Bitslice padding value	Specifies the padding value used to prepare bitslice configuration bitstreams, such that all bitslice configuration chains simultaneously receive their final configuration data bit. Default value is 1. Valid setting is 0 or 1. Use only in 2, 4, and 8-bit PS configuration mode, when you use an EPC device with the decompression feature enabled. Applies to all FPGA devices that support enhanced configuration devices.

The following table lists the symptoms you may encounter if a configuration fails, and describes the advanced options you must use to debug your configuration.

Failure Symptoms	Disable EPCS/EPCQ ID Check	Disable AS Mode CONF_DONE Error Check	PLC Settings	Post-Chain Bitstream Pad Bytes	Post-Device Bitstream Pad Bytes	Bitslice Padding Value
Configuration failure occurs after a configuration cycle.	—	Yes	Yes	Yes [9]	Yes [10]	—
Decompression feature is enabled.	—	Yes	Yes	Yes [9]	Yes [10]	—
Encryption feature is enabled.	—	Yes	Yes	Yes [9]	Yes [10]	—
CONF_DONE stays low after a configuration cycle.	—	Yes	Yes [11]	Yes [9]	Yes [10]	—
CONF_DONE goes high momentarily after a configuration cycle.	—	Yes	Yes [12]	—	—	—
FPGA does not enter user mode even though CONF_DONE goes high.	—	—	—	Yes [9]	Yes [10]	—

continued...

9 Use only for multi-device chain

10 Use only for single-device chain

11 Start with positive offset to the PLC settings

12 Start with negative offset to the PLC settings

Failure Symptoms	Disable EPCS/EPCQ ID Check	Disable AS Mode CONF_DONE Error Check	PLC Settings	Post-Chain Bitstream Pad Bytes	Post-Device Bitstream Pad Bytes	Bitslice Padding Value
Configuration failure occurs at the beginning of a configuration cycle.	Yes	—	—	—	—	—
Newly introduced EPCS, such as EPCS128.	Yes	—	—	—	—	—
Failure in `.pof` generation for EPC device using Quartus Prime Convert Programming File Utility when the decompression feature is enabled.	—	—	—	—	—	Yes

14.7 Flash Loaders

Parallel and serial configuration devices do not support the JTAG interface. However, you can use a flash loader to program configuration devices in-system via the JTAG interface. You can use an FPGA as a bridge between the JTAG interface and the configuration device. The Quartus Prime software supports parallel and serial flash loaders.

14.8 Scripting Support

In addition to the Quartus Prime Programmer GUI, you can use the Quartus Prime command-line executable `quartus_pgm.exe` (or `quartus_pgm` in Linux) to access programmer functionality from the command line and from scripts. The programmer accepts `.pof`, `.sof`, and `.jic` programming or configuration files and `.cdf` files.

The following example shows a command that programs a device:

```
quartus_pgm –c byteblasterII –m jtag –o bpv;design.pof ←
```

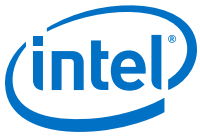
Where:

- `-c byteblasterII` specifies the Intel FPGA FPGA Parallel Port Cable download cable
- `-m jtag` specifies the JTAG programming mode
- `-o bpv` represents the blank-check, program, and verify operations
- `design.pof` represents the `.pof` used for the programming

The Programmer automatically executes the erase operation before programming the device.

For Linux terminal, use:

```
quartus_pgm –c byteblasterII –m jtag –o bpv\;design.pof
```



Related Links

About Quartus Prime Scripting
In Quartus Prime Help

14.8.1 The jtagconfig Debugging Tool

You can use the `jtagconfig` command-line utility to check the devices in a JTAG chain and the user-defined devices. The `jtagconfig` command-line utility is similar to the auto detect operation in the Quartus Prime Programmer.

For more information about the `jtagconfig` utility, use the help available at the command prompt:

```
jtagconfig [–h | --help]
```

Note: The help switch does not reference the `-n` switch. The `jtagconfig -n` command shows each node for each JTAG device.

Related Links

Command-Line Scripting
In *Quartus Prime Pro Edition Handbook Volume 2*

14.8.2 Generating a Partial-Mask SRAM Object File using a Mask Settings File and a SRAM Object File

You can generate a `.pmsf` with the `quartus_cpf` command by typing the following command:

```
quartus_cpf -p <pr_revision.msf> <pr_revision.sof> <new_filename.pmsf>
```

14.9 Document Revision History

Table 115. Document Revision History

Date	Version	Changes
2017.05.08	17.0.0	• Added Project Hash feature.
2016.10.31	16.1.0	• Implemented Intel rebranding.
2015.11.02	15.1.0	Changed instances of *Quartus II* to *Quartus Prime*.
2015.05.04	15.0.0	Added Conversion Setup File (.cof) description and example.
December 2014	14.1.0	Updated the Scripting Support section to include a Linux command to program a device.
June 2014	14.0.0	• Added Running JTAG Daemon. • Removed Cyclone III and Stratix III devices references. • Removed MegaWizard Plug-In Manager references. • Updated Secondary Programming Files section to add notes about the Quartus Prime Programmer support for **.rbf** files.

continued...

Date	Version	Changes
November 2013	13.1.0	• Converted to DITA format. • Added JTAG Debug Mode for Partial Reconfiguration and Configuring Partial Reconfiguration Bitstream in JTAG Debug Mode sections.
November 2012	12.1.0	• Updated Table 18–3 on page 18–6, and Table 18–4 on page 18–8. • Added "Converting Programming Files for Partial Reconfiguration" on page 18–10, "Generating .pmsf using a .msf and a .sof" on page 18–10, "Generating .rbf for Partial Reconfiguration Using a .pmsf" on page 18–12, "Enable Decompression during Partial Reconfiguration Option" on page 18–14 • Updated "Scripting Support" on page 18–15.
June 2012	12.0.0	• Updated Table 18–5 on page 18–8. • Updated "Quartus Prime Programmer GUI" on page 18–3.
November 2011	11.1.0	• Updated "Configuration Modes" on page 18–5. • Added "Optional Programming or Configuration Files" on page 18–6. • Updated Table 18–2 on page 18–5.
May 2011	11.0.0	• Added links to Quartus Prime Help. • Updated "Hardware Setup" on page 21–4 and "JTAG Chain Debugger Tool" on page 21–4.
December 2010	10.1.0	• Changed to new document template. • Updated "JTAG Chain Debugger Example" on page 20–4. • Added links to Quartus Prime Help. • Reorganized chapter.
July 2010	10.0.0	• Added links to Quartus Prime Help. • Deleted screen shots.
November 2009	9.1.0	No change to content.
March 2009	9.0.0	• Added a row to Table 21–4. • Changed references from "JTAG Chain Debug" to "JTAG Chain Debugger". • Updated figures.

Related Links

Altera Documentation Archive

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

15 Aldec Active-HDL and Riviera-PRO Support

You can integrate your supported EDA simulator into the Quartus Prime design flow. This chapter provides specific guidelines for simulation of Quartus Prime designs with the Aldec Active-HDL or Riviera-PRO software.

15.1 Quick Start Example (Active-HDL VHDL)

You can adapt the following RTL simulation example to get started quickly with Active-HDL:

1. Type the following to specify your EDA simulator and executable path in the Quartus Prime software:

   ```
   set_user_option -name EDA_TOOL_PATH_ACTIVEHDL <Active HDL executable path>
   ```

   ```
   set_global_assignment -name EDA_SIMULATION_TOOL "Active-HDL (VHDL)"
   ```

2. Compile simulation model libraries using one of the following methods:
   - Use Quartus Prime Simulation Library Compiler to automatically compile all required simulation models for your design.
   - Compile Intel FPGA simulation models manually:

     ```
     vlib <library1> <altera_library1>
     vcom -strict93 -dbg -work <library1> <lib1_component/pack.vhd> <lib1.vhd>
     ```

3. Create and open the workspace:

   ```
   createdesign <workspace name> <workspace path>
   opendesign -a <workspace name>.adf
   ```

4. Create the work library and compile the netlist and testbench files:

   ```
   vlib work
   vcom  -strict93  -dbg -work work <output netlist> <testbench file>
   ```

5. Load the design:

   ```
   vsim +access+r -t 1ps +transport_int_delays +transport_path_delays \
   -L work -L <lib1> -L <lib2> work.<testbench module name>
   ```

6. Run the simulation in the Active-HDL simulator.

15.2 Aldec Active-HDL and Riviera-PRO Guidelines

The following guidelines apply to simulating Intel FPGA designs in the Active-HDL or Riviera-PRO software.

15.2.1 Compiling SystemVerilog Files

If your design includes multiple SystemVerilog files, you must compile the System Verilog files together with a single alog command. If you have Verilog files and SystemVerilog files in your design, you must first compile the Verilog files, and then compile only the SystemVerilog files in the single `alog` command.

15.2.2 Simulating Transport Delays

By default, the Active-HDL or Riviera-PRO software filters out all pulses that are shorter than the propagation delay between primitives. Turning on the **transport delay** options in the in the Active-HDL or Riviera-PRO software prevents the simulator from filtering out these pulses.

Table 116. Transport Delay Simulation Options

Option	Description
`+transport_path_delays`	Use when simulation pulses are shorter than the delay in a gate-level primitive. You must include the `+pulse_e/number` and `+pulse_r/number` options.
`+transport_int_delays`	Use when simulation pulses are shorter than the interconnect delay between gate-level primitives. You must include the `+pulse_int_e/number` and `+pulse_int_r/number` options.

To perform a gate-level timing simulation with the device family library, type the Active-HDL command:

```
vsim -t 1ps -L stratixii -sdftyp /i1=filtref_vhd.sdo \
work.filtref_vhd_vec_tst +transport_int_delays +transport_path_delays
```

15.2.3 Disabling Timing Violation on Registers

In certain situations, you may want to ignore timing violations on registers and disable the "X" propagation that occurs. For example, this technique may be helpful to eliminate timing violations in internal synchronization registers in asynchronous clock-domain crossing.

By default, the **x_on_violation_option** logic option is enabled for all design registers, resulting in an output of "X" at timing violation. To disable "X" propagation at timing violations on a specific register, disable the **x_on_violation_option** logic option for the specific register, as shown in the following example from the Quartus Prime Settings File (`.qsf`).

```
set_instance_assignment -name X_ON_VIOLATION_OPTION OFF -to \
<register_name>
```

15.3 Using Simulation Setup Scripts

The Quartus Prime software can generate a rivierapro_setup.tcl simulation setup script for IP cores in your design. The use and content of the script file is similar to the **msim_setup.tcl** file used by the ModelSim simulator.

15.4 Document Revision History

Table 117. Document Revision History

Date	Version	Changes
2017.05.08	17.0.0	• Removed note about unsupported NativeLink gate level simulation.
2016.10.31	16.1.0	• Implemented Intel rebranding.
2016.05.02	16.0.0	• Removed support for NativeLink simulation in Pro Edition
2015.11.02	15.1.0	Changed instances of *Quartus II* to *Quartus Prime*.
2014.06.30	14.0.0	• Replaced MegaWizard Plug-In Manager information with IP Catalog.
November 2012	12.1.0	• Relocated general simulation information to Simulating Altera Designs.
June 2012	12.0.0	• Removed survey link.
November 2011	11.0.1	• Changed to new document template.

Related Links

Altera Documentation Archive

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

16 Synopsys VCS and VCS MX Support

You can integrate your supported EDA simulator into the Quartus Prime design flow. This document provides guidelines for simulation of Quartus Prime designs with the Synopsys VCS or VCS MX software.

16.1 Quick Start Example (VCS with Verilog)

You can adapt the following RTL simulation example to get started quickly with VCS:

1. Type the following to specify your EDA simulator and executable path in the Quartus Prime software:

   `set_user_option -name EDA_TOOL_PATH_VCS <`*VCS executable path*`>`

   `set_global_assignment -name EDA_SIMULATION_TOOL "VCS"`

2. Compile simulation model libraries using one of the following methods:
   - Use Quartus Prime Simulation Library Compiler to automatically compile all required simulation models for your design.
3. Modify the `simlib_comp.vcs` file to specify your design and testbench files.
4. Type the following to run the VCS simulator:

```
vcs -R -file simlib_comp.vcs
```

16.2 VCS and QuestaSim Guidelines

The following guidelines apply to simulation of Intel FPGA designs in the VCS or VCS MX software:

- Do not specify the -v option for `altera_lnsim.sv` because it defines a systemverilog package.
- Add `-verilog` and `+verilog2001ext+.v` options to make sure all **.v** files are compiled as verilog 2001 files, and all other files are compiled as systemverilog files.
- Add the `-lca` option for Stratix V and later families because they include IEEE-encrypted simulation files for VCS and VCS MX.
- Add `-timescale=1ps/1ps` to ensure picosecond resolution.

16.2.1 Simulating Transport Delays

By default, the VCS and VCS MX software filter out all pulses that are shorter than the propagation delay between primitives. Turning on the **transport delay** options in the VCS and VCS MX software prevents the simulator from filtering out these pulses.

Table 118. Transport Delay Simulation Options (VCS and VCS MX)

Option	Description
`+transport_path_delays`	Use when simulation pulses are shorter than the delay in a gate-level primitive. You must include the `+pulse_e/number` and `+pulse_r/number` options.
`+transport_int_delays`	Use when simulation pulses are shorter than the interconnect delay between gate-level primitives. You must include the `+pulse_int_e/number` and `+pulse_int_r/number` options.

The following VCS and VCS MX software command runs a post-synthesis simulation:

```
vcs -R <testbench>.v <gate-level netlist>.v -v <Intel FPGA device family \
library>.v +transport_int_delays +pulse_int_e/0 +pulse_int_r/0 \
+transport_path_delays +pulse_e/0 +pulse_r/0
```

16.2.2 Disabling Timing Violation on Registers

In certain situations, you may want to ignore timing violations on registers and disable the "X" propagation that occurs. For example, this technique may be helpful to eliminate timing violations in internal synchronization registers in asynchronous clock-domain crossing.

By default, the **x_on_violation_option** logic option is enabled for all design registers, resulting in an output of "X" at timing violation. To disable "X" propagation at timing violations on a specific register, disable the **x_on_violation_option** logic option for the specific register, as shown in the following example from the Quartus Prime Settings File (`.qsf`).

```
set_instance_assignment -name X_ON_VIOLATION_OPTION OFF -to \
<register_name>
```

16.3 VCS Simulation Setup Script Example

The Quartus Prime software can generate a simulation setup script for IP cores in your design. The scripts contain shell commands that compile the required simulation models in the correct order, elaborate the top-level design, and run the simulation for 100 time units by default. You can run these scripts from a Linux command shell.

The scripts for VCS and VCS MX are **vcs_setup.sh** (for Verilog HDL or SystemVerilog) and **vcsmx_setup.sh** (combined Verilog HDL and SystemVerilog with VHDL). Read the generated **.sh** script to see the variables that are available for override when sourcing the script or redefining directly if you edit the script. To set up the simulation for a design, use the command-line to pass variable values to the shell script.

Example 31. Using Command-line to Pass Simulation Variables

```
sh vcsmx_setup.sh\
USER_DEFINED_ELAB_OPTIONS=+rad\
USER_DEFINED_SIM_OPTIONS=+vcs+lic+wait
```

Example 32. Example Top-Level Simulation Shell Script for VCS-MX

```
# Run generated script to compile libraries and IP simulation files
# Skip elaboration and simulation of the IP variation
sh ./ip_top_sim/synopsys/vcsmx/vcsmx_setup.sh SKIP_ELAB=1 SKIP_SIM=1
QSYS_SIMDIR="./ip_top_sim"
#Compile top-level testbench that instantiates IP
vlogan -sverilog ./top_testbench.sv
#Elaborate and simulate the top-level design
vcs –lca –t ps <elaboration control options> top_testbench
simv <simulation control options>
```

Example 33. Example Top-Level Simulation Shell Script for VCS

```
# Run script to compile libraries and IP simulation files
sh ./ip_top_sim/synopsys/vcs/vcs_setup.sh TOP_LEVEL_NAME=”top_testbench”\
# Pass VCS elaboration options to compile files and elaborate top-level
 passed to the script as the TOP_LEVEL_NAME
USER_DEFINED_ELAB_OPTIONS="top_testbench.sv"\
# Pass in simulation options and run the simulation for specified amount of time.
USER_DEFINED_SIM_OPTIONS=”<simulation control options>
```

16.4 Document Revision History

Table 119. Document Revision History

Date	Version	Changes
2017.05.08	17.0.0	• Removed note about unsupported NativeLink gate level simulation.
2016.10.31	16.1.0	• Implemented Intel rebranding. • Removed support for `.vcd` file generation.
2016.05.02	16.0.0	• Removed support for NativeLink simulation in Pro Edition
2015.11.02	15.1.0	Changed instances of *Quartus II* to *Quartus Prime*.
2014.06.30	14.0.0	• Replaced MegaWizard Plug-In Manager information with IP Catalog.
November 2012	12.1.0	• Relocated general simulation information to Simulating Altera Designs.
June 2012	12.0.0	• Removed survey link.
November 2011	11.0.1	• Changed to new document template.

Related Links

Altera Documentation Archive

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

17 Mentor Graphics ModelSim and QuestaSim Support

You can integrate a supported EDA simulator into the Quartus Prime design flow. This document provides guidelines for simulation of designs with ModelSim-Intel FPGA Edition, ModelSim, or QuestaSim software. The entry-level ModelSim-Intel FPGA Edition software, includes precompiled simulation libraries.

Note: The latest version of the ModelSim-Intel FPGA Edition software supports native, mixed-language (VHDL/Verilog HDL/SystemVerilog) co-simulation of plain text HDL. If you have a VHDL-only simulator, you can use the ModelSim-Intel FPGA Edition software to simulate Verilog HDL modules and IP cores. Alternatively, you can purchase separate co-simulation software.

Related Links

- Simulating Designs on page 10
- Managing Quartus Prime Projects

17.1 Quick Start Example (ModelSim with Verilog)

You can adapt the following RTL simulation example to get started quickly with ModelSim:

1. Type the following to specify your EDA simulator and executable path in the Quartus Prime software:

   ```
   set_user_option -name EDA_TOOL_PATH_MODELSIM <modelsim executable path>
   ```

   ```
   set_global_assignment -name EDA_SIMULATION_TOOL "MODELSIM (verilog)"
   ```

2. Compile simulation model libraries using one of the following methods:

- Use Quartus Prime Simulation Library Compiler to automatically compile all required simulation models for your design.
- Type the following commands to create and map Intel FPGA simulation libraries manually, and then compile the models manually:

```
vlib <lib1>_ver
vmap <lib1>_ver <lib1>_ver
vlog -work <lib1> <lib1>
```

3. Compile your design and testbench files:

```
vlog -work work <design or testbench name>.v
```

4. Load the design:

```
vsim -L work -L <lib1>_ver -L <lib2>_ver work.<testbench name>
```

17.2 ModelSim, ModelSim-Intel FPGA Edition, and QuestaSim Guidelines

The following guidelines apply to simulation of designs in the ModelSim, ModelSim-Intel FPGA Edition, or QuestaSim software.

17.2.1 Using ModelSim-Intel FPGA Edition Precompiled Libraries

Precompiled libraries for both functional and gate-level simulations are provided for the ModelSim-Intel FPGA Edition software. You should not compile these library files before running a simulation. No precompiled libraries are provided for ModelSim or QuestaSim. You must compile the necessary libraries to perform functional or gate-level simulation with these tools.

The precompiled libraries provided in *<install path>*`/altera/` must be compatible with the version of the Quartus Prime software that creates the simulation netlist. To verify compatibility of precompiled libraries with your version of the Quartus Prime software, refer to the *<install path>*`/altera/version.txt` file. This file indicates the Quartus Prime software version and build of the precompiled libraries.

Note: Encrypted simulation model files shipped with the Quartus Prime software version 10.1 and later can only be read by ModelSim-Intel FPGA Edition software version 6.6c and later. These encrypted simulation model files are located at the *<Quartus Prime System directory>*`/quartus/eda/sim_lib/`*<mentor>* directory.

17.2.2 Disabling Timing Violation on Registers

In certain situations, you may want to ignore timing violations on registers and disable the "X" propagation that occurs. For example, this technique may be helpful to eliminate timing violations in internal synchronization registers in asynchronous clock-domain crossing.

By default, the **x_on_violation_option** logic option is enabled for all design registers, resulting in an output of "X" at timing violation. To disable "X" propagation at timing violations on a specific register, disable the **x_on_violation_option** logic option for the specific register, as shown in the following example from the Quartus Prime Settings File (`.qsf`).

```
set_instance_assignment -name X_ON_VIOLATION_OPTION OFF -to \
<register_name>
```

17.2.3 Passing Parameter Information from Verilog HDL to VHDL

You must use in-line parameters to pass values from Verilog HDL to VHDL.

By default, the **x_on_violation_option** logic option is enabled for all design registers, resulting in an output of "X" at timing violation. To disable "X" propagation at timing violations on a specific register, disable the **x_on_violation_option** logic option for the specific register, as shown in the following example from the Quartus Prime Settings File (`.qsf`).

```
set_instance_assignment -name X_ON_VIOLATION_OPTION OFF -to \
<register_name>
```

Example 34. In-line Parameter Passing Example

```
lpm_add_sub#(.lpm_width(12), .lpm_direction("Add"),
.lpm_type("LPM_ADD_SUB"),
.lpm_hint("ONE_INPUT_IS_CONSTANT=NO,CIN_USED=NO" ))

lpm_add_sub_component (
        .dataa (dataa),
        .datab (datab),
        .result (sub_wire0)
);
```

Note: The sequence of the parameters depends on the sequence of the GENERIC in the VHDL component declaration.

17.2.4 Increasing Simulation Speed

By default, the ModelSim and QuestaSim software runs in a debug-optimized mode.

To run the ModelSim and QuestaSim software in speed-optimized mode, add the following two vlog command-line switches. In this mode, module boundaries are flattened and loops are optimized, which eliminates levels of debugging hierarchy and may result in faster simulation. This switch is not supported in the ModelSim-Intel FPGA Edition simulator.

```
vlog -fast -05
```

17.2.5 Simulating Transport Delays

By default, the ModelSim and QuestaSim software filter out all pulses that are shorter than the propagation delay between primitives.

Turning on the **transport delay** options in the ModelSim and QuestaSim software prevents the simulator from filtering out these pulses.

Table 120. Transport Delay Simulation Options (ModelSim and QuestaSim)

Option	Description
`+transport_path_delays`	Use when simulation pulses are shorter than the delay in a gate-level primitive. You must include the `+pulse_e/number` and `+pulse_r/number` options.
`+transport_int_delays`	Use when simulation pulses are shorter than the interconnect delay between gate-level primitives. You must include the `+pulse_int_e/number` and `+pulse_int_r/number` options.

The following ModelSim and QuestaSim software command shows the command line syntax to perform a gate-level timing simulation with the device family library:

```
vsim -t 1ps -L stratixii -sdftyp /i1=filtref_vhd.sdo work.filtref_vhd_vec_tst \
+transport_int_delays +transport_path_delays
```

17.2.6 Viewing Simulation Messages

ModelSim and QuestaSim error and warning messages are tagged with a vsim or vcom code. To determine the cause and resolution for a vsim or vcom error or warning, use the verror command.

For example, ModelSim may return the following error:

```
# ** Error: C:/altera_trn/DUALPORT_TRY/simulation/modelsim/DUALPORT_TRY.vho(31):
 (vcom-1136) Unknown identifier "stratixiv"
```

In this case, type the following command:

```
verror 1136
```

The following description appears:

```
# vcom Message # 1136:
# The specified name was referenced but was not found. This indicates
# that either the name specified does not exist or is not visible at
# this point in the code.
```

Note: If your design includes deep levels of hierarchy, and the **Maintain hierarchy** EDA tools option is turned on, this may result in a large number of module instances in post-fit or post-map netlist. This condition can exceed the ModelSim-Intel FPGA Edition instance limitation.

To avoid exceeding any ModelSim-Intel FPGA Edition instance limit, turn off **Maintain hierarchy** to reduce the number of modules instances to 1 in the post-fit or post-map netlist. To acces this option, click **Assignments ➤ Settings ➤ EDA Tool Settings ➤ More Settings**.

17.2.7 Viewing Simulation Waveforms

ModelSim-Intel FPGA Edition, ModelSim, and QuestaSim automatically generate a Wave Log Format File (`.wlf`) following simulation. You can use the `.wlf` to generate a waveform view.

To view a waveform from a `.wlf` through ModelSim-Intel FPGA Edition, ModelSim, or QuestaSim, perform the following steps:

1. Type `vsim` at the command line. The **ModelSim/QuestaSim** or **ModelSim-Intel FPGA Edition** dialog box appears.
2. Click **File ➤ Datasets**. The **Datasets Browser** dialog box appears.
3. Click **Open** and select your `.wlf`.
4. Click **Done**.
5. In the Object browser, select the signals that you want to observe.
6. Click **Add ➤ Wave**, and then click **Selected Signals**.
   You must first convert the `.vcd` to a `.wlf` before you can view a waveform in ModelSim-Intel FPGA Edition, ModelSim, or QuestaSim.
7. To convert the the `.vcd` to a `.wlf`, type the following at the command-line:

```
vcd2wlf <example>.vcd <example>.wlf
```

8. After conversion, view the `.wlf` waveform in ModelSim or QuestaSim.

17.2.8 Simulating with ModelSim-Intel FPGA Edition Waveform Editor

You can use the ModelSim-Intel FPGA Edition waveform editor as a simple method to create stimulus vectors for simulation. You can create this design stimulus via interactive manipulation of waveforms from the wave window in ModelSim-Intel FPGA Edition. With the ModelSim-Intel FPGA Edition waveform editor, you can create and edit waveforms, drive simulation directly from created waveforms, and save created waveforms into a stimulus file.

Related Links

ModelSim Web Page

17.3 ModelSim Simulation Setup Script Example

The Quartus Prime software can generate a msim_setup.tcl simulation setup script for IP cores in your design. The script compiles the required device library models, compiles the design files, and elaborates the design with or without simulator optimization. To run the script, type source `msim_setup.tcl` in the simulator Transcript window.

Alternatively, if you are using the simulator at the command line, you can type the following command:

```
vsim -c -do msim_setup.tcl
```

In this example the `top-level-simulate.do` custom top-level simulation script sets the hierarchy variable `TOP_LEVEL_NAME` to `top_testbench` for the design, and sets the variable `QSYS_SIMDIR` to the location of the generated simulation files.

```
# Set hierarchy variables used in the IP-generated files
set TOP_LEVEL_NAME "top_testbench"
set QSYS_SIMDIR "./ip_top_sim"
# Source generated simulation script which defines aliases used below
source $QSYS_SIMDIR/mentor/msim_setup.tcl
# dev_com alias compiles simulation libraries for device library files
dev_com
# com alias compiles IP simulation or Qsys model files and/or Qsys model files in
the correct order
com
# Compile top level testbench that instantiates your IP
vlog -sv ./top_testbench.sv
# elab alias elaborates the top-level design and testbench
elab
# Run the full simulation
run - all
```

In this example, the top-level simulation files are stored in the same directory as the original IP core, so this variable is set to the IP-generated directory structure. The `QSYS_SIMDIR` variable provides the relative hierarchy path for the generated IP simulation files. The script calls the generated `msim_setup.tcl` script and uses the alias commands from the script to compile and elaborate the IP files required for simulation along with the top-level simulation testbench. You can specify additional simulator elaboration command options when you run the `elab` command, for example, `elab +nowarnTFMPC`. The last command run in the example starts the simulation.

17.4 Unsupported Features

The Quartus Prime software does not support the following ModelSim simulation features:

- Quartus Prime does not support companion licensing for ModelSim AE.
- The USB software guard is not supported by versions earlier than Mentor Graphics ModelSim software version 5.8d.
- For ModelSim-Intel FPGA Edition software versions prior to 5.5b, use the **PCLS** utility included with the software to set up the license.
- Some versions of ModelSim and QuestaSim support SystemVerilog, PSL assertions, SystemC, and more. For more information about specific feature support, refer to Mentor Graphics literature

Related Links

ModelSim-Intel FPGA Edition Software Web Page

17.5 Document Revision History

Table 121. Document Revision History

Date	Version	Changes
2017.05.08	17.0.0	• Removed note about unsupported NativeLink gate level simulation.
2016.10.31	16.1.0	• Implemented Intel rebranding. • Corrected load design syntax error.
2016.05.02	16.0.0	• Removed support for NativeLink simulation in Pro Edition • Added note about avoiding ModelSim - Intel FPGA Edition instance limitations.
2015.11.02	15.1.0	Changed instances of *Quartus II* to *Quartus Prime*.
2015.05.04	15.0.0	• Added mixed language simulation support in the ModelSim - Intel FPGA Edition software.
2014.06.30	14.0.0	• Replaced MegaWizard Plug-In Manager information with IP Catalog.
November 2012	12.1.0	• Relocated general simulation information to Simulating Altera Designs.
June 2012	12.0.0	• Removed survey link.
November 2011	11.0.1	• Changed to new document template.

Related Links

Altera Documentation Archive

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

18 Cadence Incisive Enterprise (IES) Support

You can integrate your supported EDA simulator into the Quartus Primedesign flow. This chapter provides specific guidelines for simulation of Quartus Prime designs with the Cadence Incisive Enterprise (IES) software.

18.1 Quick Start Example (NC-Verilog)

You can adapt the following RTL simulation example to get started quickly with IES:

1. Type the following to specify your EDA simulator and executable path in the Quartus Prime software:

   ```
   set_user_option -name EDA_TOOL_PATH_NCSIM <ncsim executable path>
   ```

   ```
   set_global_assignment -name EDA_SIMULATION_TOOL "NC-Verilog (Verilog)"
   ```

2. Compile simulation model libraries using one of the following methods:
   - Use Quartus Prime Simulation Library Compiler to automatically compile all required simulation models for your design.
   - Map Intel FPGA simulation libraries by adding the following commands to a cds.lib file:

     ```
     include ${CDS_INST_DIR}/tools/inca/files/cds.lib
     DEFINE <lib1>_ver <lib1_ver>
     ```

     Then, compile Intel FPGA simulation models manually:

     ```
     vlog -work <lib1_ver>
     ```

3. Elaborate your design and testbench with IES:

   ```
   ncelab <work library>.<top-level entity name>
   ```

4. Run the simulation:

   ```
   ncsim <work library>.<top-level entity name>
   ```

18.2 Cadence Incisive Enterprise (IES) Guidelines

The following guidelines apply to simulation of Intel FPGA designs in the IES software:

- Do not specify the -v option for `altera_lnsim.sv` because it defines a systemverilog package.
- Add `-verilog` and `+verilog2001ext+.v` options to make sure all **.v** files are compiled as verilog 2001 files, and all other files are compiled as systemverilog files.
- Add the `-lca` option for Stratix V and later families because they include IEEE-encrypted simulation files for IES.
- Add `-timescale=1ps/1ps` to ensure picosecond resolution.

18.2.1 Using GUI or Command-Line Interfaces

Intel FPGA supports both the IES GUI and command-line simulator interfaces.

To start the IES GUI, type `nclaunch` at a command prompt.

Table 122. Simulation Executables

Program	Function
`ncvlog` `ncvhdl`	`ncvlog` compiles your Verilog HDL code and performs syntax and static semantics checks. `ncvhdl` compiles your VHDL code and performs syntax and static semantics checks.
`ncelab`	Elaborates the design hierarchy and determines signal connectivity.
`ncsdfc`	Performs back-annotation for simulation with VHDL simulators.
`ncsim`	Runs mixed-language simulation. This program is the simulation kernel that performs event scheduling and executes the simulation code.

18.2.2 Elaborating Your Design

The simulator automatically reads the `.sdo` file during elaboration of the Quartus Prime-generated Verilog HDL or SystemVerilog HDL netlist file. The ncelab command recognizes the embedded system task `$sdf_annotate` and automatically compiles and annotates the `.sdo` file by running `ncsdfc` automatically.

VHDL netlist files do not contain system task calls to locate your `.sdf` file; therefore, you must compile the standard `.sdo` file manually. Locate the `.sdo` file in the same directory where you run elaboration or simulation. Otherwise, the `$sdf_annotate` task cannot reference the `.sdo` file correctly. If you are starting an elaboration or simulation from a different directory, you can either comment out the `$sdf_annotate` and annotate the `.sdo` file with the GUI, or add the full path of the `.sdo` file.

Note: If you use NC-Sim for post-fit VHDL functional simulation of a Stratix V design that includes RAM, an elaboration error might occur if the component declaration parameters are not in the same order as the architecture parameters. Use the `-namemap_mixgen` option with the `ncelab` command to match the component declaration parameter and architecture parameter names.

18.2.3 Back-Annotating Simulation Timing Data (VHDL Only)

You can back annotate timing information in a Standard Delay Output File (.sdo) for VHDL simulators. To back annotate the .sdo timing data at the command line, follow these steps:

1. To compile the **.sdo** with the ncsdfc program, type the following command at the command prompt. The ncsdfc program generates an <*output name*>**.sdf.X** compiled **.sdo** file

   ```
   ncsdfc <project name>_vhd.sdo –output <output name>
   ```

   Note: If you do not specify an output name, ncsdfc uses <*project name*>**.sdo.X**
2. Specify the compiled **.sdf** file for the project by adding the following command to an ASCII SDF command file for the project:

   ```
   COMPILED_SDF_FILE = "<project name>.sdf.X" SCOPE = <instance path>
   ```
3. After compiling the **.sdf** file, type the following command to elaborate the design:

   ```
   ncelab worklib.<project name>:entity –SDF_CMD_FILE <SDF Command File>
   ```

Example 35. Example SDF Command File

```
// SDF command file sdf_file
COMPILED_SDF_FILE = "lpm_ram_dp_test_vhd.sdo.X",
SCOPE = :tb,
MTM_CONTROL = "TYPICAL",
SCALE_FACTORS = "1.0:1.0:1.0",
SCALE_TYPE = "FROM_MTM";
```

18.2.4 Disabling Timing Violation on Registers

In certain situations, you may want to ignore timing violations on registers and disable the “X” propagation that occurs. For example, this technique may be helpful to eliminate timing violations in internal synchronization registers in asynchronous clock-domain crossing.

By default, the **x_on_violation_option** logic option is enabled for all design registers, resulting in an output of “X” at timing violation. To disable “X” propagation at timing violations on a specific register, disable the **x_on_violation_option** logic option for the specific register, as shown in the following example from the Quartus Prime Settings File (.qsf).

```
set_instance_assignment -name X_ON_VIOLATION_OPTION OFF -to \
<register_name>
```

18.2.5 Simulating Pulse Reject Delays

By default, the IES software filters out all pulses that are shorter than the propagation delay between primitives.
Setting the pulse reject delays options in the IES software prevents the simulation tool from filtering out these pulses. Use the following options to ensure that all signal pulses are seen in the simulation results.

Table 123. Pulse Reject Delay Options

Program	Function
-PULSE_R	Use when simulation pulses are shorter than the delay in a gate-level primitive. The argument is the percentage of delay for pulse reject limit for the path
-PULSE_INT_R	Use when simulation pulses are shorter than the interconnect delay between gate-level primitives. The argument is the percentage of delay for pulse reject limit for the path

18.2.6 Viewing Simulation Waveforms

IES generates a `.trn` file automatically following simulation. You can use the `.trn` for generating the SimVision waveform view.

To view a waveform from a `.trn` file through SimVision, follow these steps:

1. Type `simvision` at the command line. The **Design Browser** dialog box appears.
2. Click **File ➤ Open Database** and click the `.trn` file.
3. In the **Design Browser** dialog box, select the signals that you want to observe from the Hierarchy.
4. Right-click the selected signals and click **Send to Waveform Window**.

   You cannot view a waveform from a `.vcd` file in SimVision, and the `.vcd` file cannot be converted to a `.trn` file.

18.3 IES Simulation Setup Script Example

The Quartus Prime software can generate a `ncsim_setup.sh` simulation setup script for IP cores in your design. The script contains shell commands that compile the required device libraries, IP, or Qsys Pro simulation models in the correct order. The script then elaborates the top-level design and runs the simulation for 100 time units by default. You can run these scripts from a Linux command shell. To set up the simulation script for a design, you can use the command-line to pass variable values to the shell script.

Read the generated `.sh` script to see the variables that are available for you to override when you source the script or that you can redefine directly in the generated .sh script. For example, you can specify additional elaboration and simulation options with the variables USER_DEFINED_ELAB_OPTIONS and USER_DEFINED_SIM_OPTIONS.

Example 36. Example Top-Level Simulation Shell Script for Incisive (NCSIM)

```
# Run script to compile libraries and IP simulation files
# Skip elaboration and simulation of the IP variation
sh ./ip_top_sim/cadence/ncsim_setup.sh SKIP_ELAB=1 SKIP_SIM=1 QSYS_SIMDIR="./
```

```
ip_top_sim"

#Compile the top-level testbench that instantiates your IP
ncvlog -sv ./top_testbench.sv
#Elaborate and simulate the top-level design
ncelab <elaboration control options> top_testbench
ncsim <simulation control options> top_testbench
```

18.4 Document Revision History

Table 124. Document Revision History

Date	Version	Changes
2016.10.31	16.1.0	• Implemented Intel rebranding.
2016.05.02	16.0.0	• Removed support for NativeLink simulation in Pro Edition
2015.11.02	15.1.0	Changed instances of *Quartus II* to *Quartus Prime*.
2014.08.18	14.0.a10.0	• Corrected incorrect references to VCS and VCS MX.
2014.06.30	14.0.0	• Replaced MegaWizard Plug-In Manager information with IP Catalog.
November 2012	12.1.0	• Relocated general simulation information to Simulating Altera Designs.
June 2012	12.0.0	• Removed survey link.
November 2011	11.0.1	• Changed to new document template.

Related Links

Altera Documentation Archive

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