Intel® Quartus® Prime Standard Edition User Guide

Third-party Synthesis

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

Subscribe
Send Feedback

UG-20181 | 2018.09.24
Latest document on the web: PDF | HTML

Contents

1. Synopsys Synplify* Support

1.1. About Synplify Support

the Intel® Quartus® Prime software supports use of the Synopsys Synplify software design flows, methodologies, and techniques for achieving optimal results in Intel devices. Synplify support applies to Synplify, Synplify Pro, and Synplify Premier software. This document assumes proper set up, licensing, and basic familiarity with the Synplify software.

This document covers the following information:

- General design flow with the Synplify and Intel Quartus Prime software.
- Exporting designs and constraints to the Intel Quartus Prime software.
- Synplify software optimization strategies, including timing-driven compilation settings, optimization options, and other attributes.
- Guidelines for use of Quartus Prime IP cores, including guidelines for HDL inference of IP cores.

Related Information

- Synplify Synthesis Techniques with the Intel Quartus Prime Software online training
- Synplify Pro Tips and Tricks online training

1.2. Design Flow

The following steps describe a basic Intel Quartus Prime software design flow using the Synplify software:

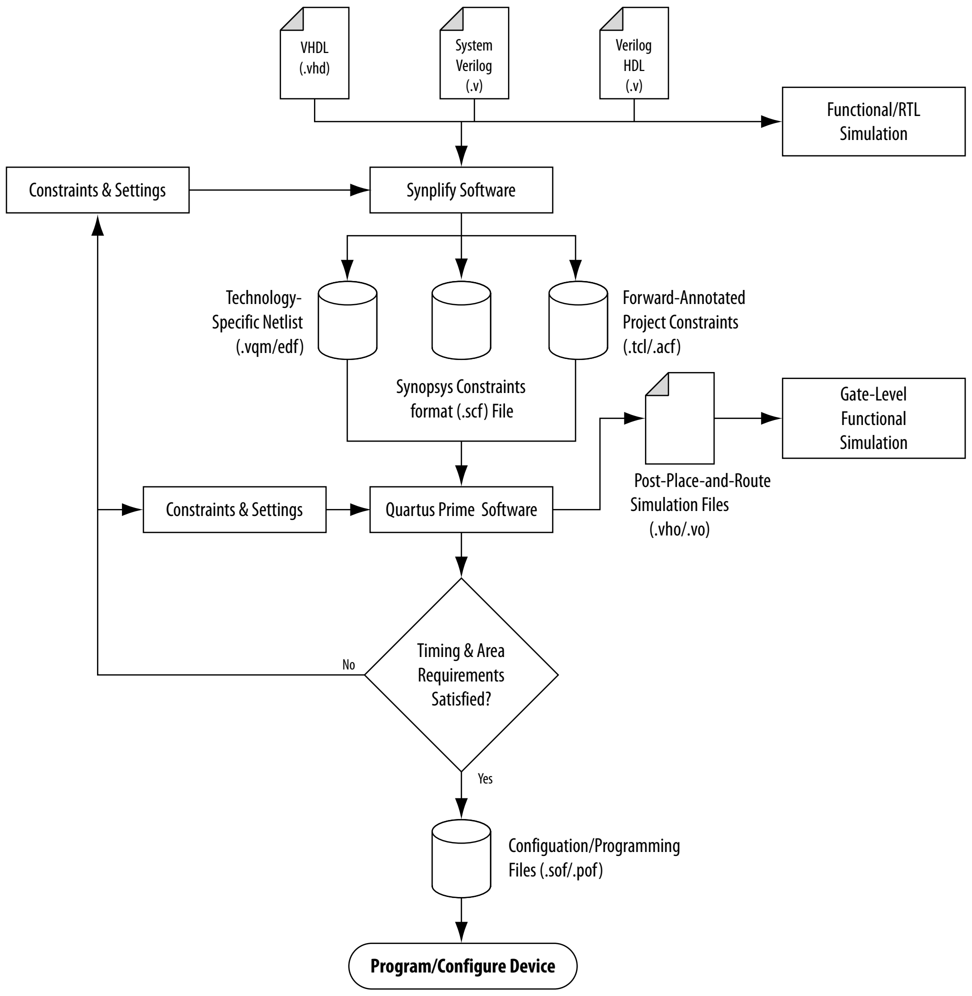
1. Create Verilog HDL (`.v`) or VHDL (`.vhd`) design files.
2. Set up a project in the Synplify software and add the HDL design files for synthesis.
3. Select a target device and add timing constraints and compiler directives in the Synplify software to help optimize the design during synthesis.
4. Synthesize the project in the Synplify software.
5. Create an Intel Quartus Prime project and import the following files generated by the Synplify software into the Intel Quartus Prime software. Use the following files for placement and routing, and for performance evaluation:

- Verilog Quartus Mapping File (`.vqm`) netlist.
- The Synopsys Constraints Format (`.scf`) file for Timing Analyzer constraints.
- The `.tcl` file to set up your Intel Quartus Prime project and pass constraints.

  Note: Alternatively, you can run the Intel Quartus Prime software from within the Synplify software.

6. After obtaining place-and-route results that meet your requirements, configure or program the Intel device.

Figure 1. Recommended Design Flow


VHDL (.vhd)
System Verilog (.v)
Verilog HDL (.v)
Functional/RTL Simulation
Constraints & Settings
Synplify Software
Technology-Specific Netlist (.vqm/edf)
Synopsys Constraints format (.scf) File
Forward-Annotated Project Constraints (.tcl/.acf)
Gate-Level Functional Simulation
Post-Place-and-Route Simulation Files (.vho/.vo)
Constraints & Settings
Quartus Prime Software
Timing & Area Requirements Satisfied?
No
Yes
Configuation/Programming Files (.sof/.pof)
Program/Configure Device

Related Information

- Running the Intel Quartus Prime Software from within the Synplify Software on page 7
- Synplify Software Generated Files on page 8
- Design Constraints Support on page 9

1.3. Hardware Description Language Support

The Synplify software supports VHDL, Verilog HDL, and SystemVerilog source files. However, only the Synplify Pro and Premier software support mixed synthesis, allowing a combination of VHDL and Verilog HDL or SystemVerilog format source files.

The HDL Analyst that is included in the Synplify software is a graphical tool for generating schematic views of the technology-independent RTL view netlist (`.srs`) and technology-view netlist (`.srm`) files. You can use the Synplify HDL Analyst to analyze and debug your design visually. The HDL Analyst supports cross-probing between the RTL and Technology views, the HDL source code, the Finite State Machine (FSM) viewer, and between the technology view and the timing report file in the Intel Quartus Prime software. A separate license file is required to enable the HDL Analyst in the Synplify software. The Synplify Pro and Premier software include the HDL Analyst.

Related Information

Guidelines for Intel FPGA IP Cores and Architecture-Specific Features on page 18

1.4. Intel Device Family Support

Support for newly released device families may require an overlay. Contact Synopsys for more information.

Related Information

Synopsys Website

1.5. Tool Setup

1.5.1. Specifying the Intel Quartus Prime Software Version

You can specify your version of the Intel Quartus Prime software in **Implementation Options** in the Synplify software. This option ensures that the netlist is compatible with the software version and supports the newest features. Intel recommends using the latest version of the Intel Quartus Prime software whenever possible. If your Intel Quartus Prime software version is newer than the versions available in the **Quartus Version** list, check if there is a newer version of the Synplify software available that supports the current Intel Quartus Prime software version. Otherwise, select the latest version in the list for the best compatibility.

Note: The **Quartus Version** list is available only after selecting an Intel device.

Example 1. Specifying Intel Quartus Prime Software Version at the Command Line

```
set_option -quartus_version <version number>
```

1.5.2. Exporting Designs to the Intel Quartus Prime Software Using NativeLink Integration

The NativeLink feature in the Intel Quartus Prime software facilitates the seamless transfer of information between the Intel Quartus Prime software and EDA tools, and allows you to run other EDA design entry or synthesis, simulation, and timing analysis tools automatically from within the Intel Quartus Prime software. After a design is synthesized in the Synplify software, a `.vqm` netlist file, an `.scf` file for Timing Analyzer timing constraints, and `.tcl` files are used to import the design into the Intel Quartus Prime software for place-and-route. You can run the Intel Quartus Prime software from within the Synplify software or as a stand-alone application. After you import the design into the Intel Quartus Prime software, you can specify different options to further optimize the design.

Note: When you are using NativeLink integration, the path to your project must not contain empty spaces. The Synplify software uses Tcl scripts to communicate with the Intel Quartus Prime software, and the Tcl language does not accept arguments with empty spaces in the path.

Use NativeLink integration to integrate the Synplify software and Intel Quartus Prime software with a single GUI for both synthesis and place and-route operations. NativeLink integration allows you to run the Intel Quartus Prime software from within the Synplify software GUI, or to run the Synplify software from within the Intel Quartus Prime software GUI.

1.5.2.1. Running the Intel Quartus Prime Software from within the Synplify Software

To run the Intel Quartus Prime software from within the Synplify software, you must set the *QUARTUS_ROOTDIR* environment variable to the Intel Quartus Prime software installation directory located in <*Intel Quartus Prime system directory*>\altera\ <*version number*>**\quartus**. You must set this environment variable to use the Synplify and Intel Quartus Prime software together. Synplify also uses this variable to open the Intel Quartus Prime software in the background and obtain detailed information about the Intel FPGA IP cores used in the design.

For the Windows operating system, do the following:

1. Point to **Start**, and click **Control Panel**.
2. Click **System** >**Advanced system settings** >**Environment Variables**.
3. Create a *QUARTUS_ROOTDIR* system variable.

For the Linux operating system, do the following:

- Create an environment variable *QUARTUS_ROOTDIR* that points to the <*home directory*>/altera <*version number*> location.

You can create new place and route implementations with the **New P&R** button in the Synplify software GUI. Under each implementation, the Synplify Pro software creates a place-and-route implementation called **pr_**<*number*> **Altera Place and Route**. To run the Intel Quartus Prime software in command-line mode after each synthesis run, use the text box to turn on the place-and-route implementation. The results of the place-and-route are written to a log file in the **pr_** <*number*> directory under the current implementation directory.

You can also use the commands in the Intel Quartus Prime menu to run the Intel Quartus Prime software at any time following a successful completion of synthesis. In the Synplify software, on the Options menu, click **Intel Quartus Prime** and then choose one of the following commands:

- **Launch Quartus** —Opens the Intel Quartus Prime software GUI and creates a Intel Quartus Prime project with the synthesized output file, forward-annotated timing constraints, and pin assignments. Use this command to configure options for the project and to execute any Intel Quartus Prime commands.
- **Run Background Compile**—Runs the Intel Quartus Prime software in command-line mode with the project settings from the synthesis run. The results of the place-and-route are written to a log file.

The <*project_name*>**_cons.tcl** file is used to set up the Intel Quartus Prime project and directs the <*project_name*>**.tcl** file to pass constraints from the Synplify software to the Intel Quartus Prime software. By default, the <*project_name*>**.tcl** file contains device, timing, and location assignments. The <*project_name*>**.tcl** file contains the command to use the Synplify-generated **.scf** constraints file with the Timing Analyzer.

Related Information

Design Flow on page 4

1.5.2.2. Using the Intel Quartus Prime Software to Run the Synplify Software

You can set up the Intel Quartus Prime software to run the Synplify software for synthesis with NativeLink integration. This feature allows you to use the Synplify software to quickly synthesize a design as part of a standard compilation in the Intel Quartus Prime software. When you use this feature, the Synplify software does not use any timing constraints or assignments that you have set in the Intel Quartus Prime software.

Note: For best results, Synopsys recommends that you set constraints in the Synplify software and use a Tcl script to pass these constraints to the Intel Quartus Prime software, instead of opening the Synplify software from within the Intel Quartus Prime software.

To set up the Intel Quartus Prime software to run the Synplify software, do the following:

1. On the Tools menu, click **Options**.
2. In the **Options** dialog box, click **EDA Tool Options** and specify the path of the Synplify or Synplify Pro software under **Location of Executable**.

Running the Synplify software with NativeLink integration is supported on both floating network and node-locked fixed PC licenses. Both types of licenses support batch mode compilation.

1.6. Synplify Software Generated Files

During synthesis, the Synplify software produces several intermediate and output files.

Send Feedback

Table 1. Synplify Intermediate and Output Files

File Extensions	File Description
`.vqm`	Technology-specific netlist in `.vqm` file format. A `.vqm` file is created for all Intel device families supported by the Intel Quartus Prime software.
`.scf`(1)	Synopsys Constraint Format file containing timing constraints for the Timing Analyzer.
`.tcl`	Forward-annotated constraints file containing constraints and assignments. A `.tcl` file for the Intel Quartus Prime software is created for all devices. The `.tcl`file contains the appropriate Tcl commands to create and set up an Intel Quartus Prime project and pass placement constraints.
`.srs`	Technology-independent RTL netlist file that can be read only by the Synplify software.
`.srm`	Technology view netlist file.
`.acf`	Assignment and Configurations file for backward compatibility with the MAX+PLUS II software. For devices supported by the MAX+PLUS II software, the MAX+PLUS II assignments are imported from the MAX+PLUS II `.acf` file.
`.srr`(2)	Synthesis Report file.

Related Information

Design Flow on page 4

1.7. Design Constraints Support

You can specify timing constraints and attributes by using the SCOPE window of the Synplify software, by editing the `.sdc` file, or by defining the compiler directives in the HDL source file. The Synplify software forward-annotates many of these constraints to the Intel Quartus Prime software.

After synthesis is complete, do the following steps:

1. Import the `.vqm` netlist to the Intel Quartus Prime software for place-and-route.
2. Use the `.tcl` file generated by the Synplify software to forward-annotate your project constraints including device selection. The `.tcl` file calls the generated `.scf` to forward-annotate Timing Analyzer timing constraints.

(1) If your design uses the Classic Timing Analyzer for timing analysis in the Intel Quartus Prime software versions 10.0 and earlier, the Synplify software generates timing constraints in the Tcl Constraints File (`.tcl`). If you are using the Intel Quartus Prime software versions 10.1 and later, you must use the Timing Analyzer for timing analysis.

(2) This report file includes performance estimates that are often based on pre-place-and-route information. Use the f_{MAX} reported by the Intel Quartus Prime software after place-and-route—it is the only reliable source of timing information. This report file includes post-synthesis device resource utilization statistics that might inaccurately predict resource usage after place-and-route. The Synplify software does not account for black box functions nor for logic usage reduction achieved through register packing performed by the Intel Quartus Prime software. Register packing combines a single register and look-up table (LUT) into a single logic cell, reducing logic cell utilization below the Synplify software estimate. Use the device utilization reported by the Intel Quartus Prime software after place-and-route.

Related Information

- Design Flow on page 4
- Synplify Optimization Strategies on page 11

1.7.1. Running the Intel Quartus Prime Software Manually With the Synplify-Generated Tcl Script

You can run the Intel Quartus Prime software with a Synplify-generated Tcl script.

To run the Tcl script to set up your project assignments, perform the following steps:

1. Ensure the `.vqm`, `.scf`, and `.tcl` files are located in the same directory.
2. In the Intel Quartus Prime software, on the View menu, point to **Utility Windows** and click **Tcl Console**. The Intel Quartus Prime Tcl Console opens.
3. At the Tcl Console command prompt, type the following:

```
source <path>/<project name>_cons.tcl
```

1.7.2. Passing Timing Analyzer SDC Timing Constraints to the Intel Quartus Prime Software

The Timing Analyzer is a powerful ASIC-style timing analysis tool that validates the timing performance of all logic in your design using an industry standard constraints format, Synopsys Design Constraints (`.sdc`).

The Synplify-generated `.tcl` file contains constraints for the Intel Quartus Prime software, such as the device specification and any location constraints. Timing constraints are forward-annotated in the Synopsys Constraints Format (`.scf`) file.

Note: Synopsys recommends that you modify constraints using the SCOPE constraint editor window, rather than using the generated `.sdc`, `.scf`, or `.tcl` file.

The following list of Synplify constraints are converted to the equivalent Intel Quartus Prime SDC commands and are forward-annotated to the Intel Quartus Prime software in the `.scf`file:

- `define_clock`
- `define_input_delay`
- `define_output_delay`
- `define_multicycle_path`
- `define_false_path`

All Synplify constraints described above are mapped to SDC commands for the Timing Analyzer.

For syntax and arguments for these commands, refer to the applicable topic in this manual or refer to Synplify Help. For a list of corresponding commands in the Intel Quartus Prime software, refer to the Intel Quartus Prime Help.

Related Information

Timing-Driven Synthesis Settings on page 12

1.7.2.1. Individual Clocks and Frequencies

Specify clock frequencies for individual clocks in the Synplify software with the `define_clock` command. This command is passed to the Intel Quartus Prime software with the `create_clock` command.

1.7.2.2. Input and Output Delay

Specify input delay and output delay constraints in the Synplify software with the `define_input_delay` and `define_output_delay` commands, respectively. These commands are passed to the Intel Quartus Prime software with the `set_input_delay` and `set_output_delay` commands.

1.7.2.3. Multicycle Path

Specify a multicycle path constraint in the Synplify software with the `define_multicycle_path` command. This command is passed to the Intel Quartus Prime software with the `set_multicycle_path` command.

1.7.2.4. False Path

Specify a false path constraint in the Synplify software with the `define_false_path` command. This command is passed to the Intel Quartus Prime software with the `set_false_path` command.

1.8. Simulation and Formal Verification

You can perform simulation and formal verification at various stages in the design process. You can perform final timing analysis after placement and routing is complete.

If area and timing requirements are satisfied, use the files generated by the Intel Quartus Prime software to program or configure the Intel device. If your area or timing requirements are not met, you can change the constraints in the Synplify software or the Intel Quartus Prime software and rerun synthesis. Intel recommends that you provide timing constraints in the Synplify software and any placement constraints in the Intel Quartus Prime software. Repeat the process until area and timing requirements are met.

You can also use other options and techniques in the Intel Quartus Prime software to meet area and timing requirements, such as WYSIWYG Primitive Resynthesis, which can perform optimizations on your `.vqm` netlist within the Intel Quartus Prime software.

Note: In some cases, you might be required to modify the source code if the area and timing requirements cannot be met using options in the Synplify and Intel Quartus Prime software.

1.9. Synplify Optimization Strategies

Combining Synplify software constraints with VHDL and Verilog HDL coding techniques and Intel Quartus Prime software options can help you obtain the results that you require.

For more information about applying attributes, refer to the *Synopsys FPGA Synthesis Reference Manual*.

Related Information

Design Constraints Support on page 9

1.9.1. Using Synplify Premier to Optimize Your Design

Compared to other Synplify products, the Synplify Premier software offers additional physical synthesis optimizations. After typical logic synthesis, the Synplify Premier software places and routes the design and attempts to restructure the netlist based on the physical location of the logic in the Intel device. The Synplify Premier software forward-annotates the design netlist to the Intel Quartus Prime software to perform the final placement and routing. In the default flow, the Synplify Premier software also forward-annotates placement information for the critical path(s) in the design, which can improve the compilation time in the Intel Quartus Prime software.

The physical location annotation file is called <*design name*>`_plc.tcl`. If you open the Intel Quartus Prime software from the Synplify Premier software user interface, the Intel Quartus Prime software automatically uses this file for the placement information.

The Physical Analyst allows you to examine the placed netlist from the Synplify Premier software, which is similar to the HDL Analyst for a logical netlist. You can use this display to analyze and diagnose potential problems.

1.9.2. Using Implementations in Synplify Pro or Premier

You can create different synthesis results without overwriting the existing results, in the Synplify Pro or Premier software, by creating a new implementation from the Project menu. For each implementation, specify the target device, synthesis options, and constraint files. Each implementation generates its own subdirectory that contains all the resulting files, including `.vqm`, `.scf`, and `.tcl` files, from a compilation of the particular implementation. You can then compare the results of the different implementations to find the optimal set of synthesis options and constraints for a design.

1.9.3. Timing-Driven Synthesis Settings

The Synplify software supports timing-driven synthesis with user-assigned timing constraints to optimize the performance of the design.

The Intel Quartus Prime NativeLink feature allows timing constraints that are applied in the Synplify software to be forward-annotated for the Intel Quartus Prime software with an `.scf` file for timing-driven place and route.

The Synplify Synthesis Report File (`.srr`) contains timing reports of estimated place-and-route delays. The Intel Quartus Prime software can perform further optimizations on a post-synthesis netlist from third-party synthesis tools. In addition, designs might contain black boxes or intellectual property (IP) functions that have not been optimized by the third-party synthesis software. Actual timing results are obtained only after the design has been fully placed and routed in the Intel Quartus Prime software. For these reasons, the Intel Quartus Prime post place-and-route timing reports provide a more accurate representation of the design. Use the statistics in these reports to evaluate design performance.

Related Information

Passing Timing Analyzer SDC Timing Constraints to the Intel Quartus Prime Software on page 10

1.9.3.1. Clock Frequencies

For single-clock designs, you can specify a global frequency when using the push-button flow. While this flow is simple and provides good results, it often does not meet the performance requirements for more advanced designs. You can use timing constraints, compiler directives, and other attributes to help optimize the performance of a design. You can enter these attributes and directives directly in the HDL code. Alternatively, you can enter attributes (not directives) into an `.sdc` file with the SCOPE window in the Synplify software.

Use the SCOPE window to set global frequency requirements for the entire design and individual clock settings. Use the **Clocks** tab in the SCOPE window to specify frequency (or period), rise times, fall times, duty cycle, and other settings. Assigning individual clock settings, rather than over-constraining the global frequency, helps the Intel Quartus Prime software and the Synplify software achieve the fastest clock frequency for the overall design. The `define_clock` attribute assigns clock constraints.

1.9.3.2. Multiple Clock Domains

The Synplify software can perform timing analysis on unrelated clock domains. Each clock group is a different clock domain and is treated as unrelated to the clocks in all other clock groups. All clocks in a single clock group are assumed to be related, and the Synplify software automatically calculates the relationship between the clocks. You can assign clocks to a new clock group or put related clocks in the same clock group with the **Clocks** tab in the SCOPE window, or with the `define_clock` attribute.

1.9.3.3. Input and Output Delays

Specify the input and output delays for the ports of a design in the **Input/Output** tab of the SCOPE window, or with the `define_input_delay` and `define_output_delay` attributes. The Synplify software does not allow you to assign the t_{CO} and t_{SU} values directly to inputs and outputs. However, a t_{CO} value can be inferred by setting an external output delay; a t_{SU} value can be inferred by setting an external input delay.

Relationship Between t_{CO} and the Output Delay
t_{CO} = clock period – external output delay

Relationship Between t_{SU} and the Input Delay
t_{SU} = clock period – external input delay

When the `syn_forward_io_constraints` attribute is set to 1, the Synplify software passes the external input and output delays to the Intel Quartus Prime software using NativeLink integration. The Intel Quartus Prime software then uses the external delays to calculate the maximum system frequency.

1.9.3.4. Multicycle Paths

A multicycle path is a path that requires more than one clock cycle to propagate. Specify any multicycle paths in the design in the **Multi-Cycle Paths** tab of the SCOPE window, or with the `define_multicycle_path` attribute. You should specify which paths are multicycle to prevent the Intel Quartus Prime and the Synplify compilers from working excessively on a non-critical path. Not specifying these paths can also result in an inaccurate critical path reported during timing analysis.

1.9.3.5. False Paths

False paths are paths that should be ignored during timing analysis, or should be assigned low (or no) priority during optimization. Some examples of false paths include slow asynchronous resets, and test logic that has been added to the design. Set these paths in the **False Paths** tab of the SCOPE window, or use the `define_false_path` attribute.

1.9.4. FSM Compiler

If the FSM Compiler is turned on, the compiler automatically detects state machines in a design, which are then extracted and optimized. The FSM Compiler analyzes state machines and implements sequential, gray, or one-hot encoding, based on the number of states. The compiler also performs unused-state analysis, optimization of unreachable states, and minimization of transition logic. Implementation is based on the number of states, regardless of the coding style in the HDL code.

If the FSM Compiler is turned off, the compiler does not optimize logic as state machines. The state machines are implemented as HDL code. Thus, if the coding style for a state machine is sequential, the implementation is also sequential.

Use the `syn_state_machine` compiler directive to specify or prevent a state machine from being extracted and optimized. To override the default encoding of the FSM Compiler, use the `syn_encoding` directive.

Table 2. `syn_encoding` Directive Values

Value	Description
Sequential	Generates state machines with the fewest possible flipflops. Sequential, also called binary, state machines are useful for area-critical designs when timing is not the primary concern.
Gray	Generates state machines where only one flipflop changes during each transition. Gray-encoded state machines tend to be glitches.
One-hot	Generates state machines containing one flipflop for each state. One-hot state machines typically provide the best performance and shortest clock-to-output delays. However, one-hot implementations are usually larger than sequential implementations.
Safe	Generates extra control logic to force the state machine to the reset state if an invalid state is reached. You can use the safe value in conjunction with any of the other three values, which results in the state machine being implemented with the requested encoding scheme and the generation of the reset logic.

Example 2. Sample VHDL Code for Applying syn_encoding Directive

```
SIGNAL current_state : STD_LOGIC_VECTOR (7 DOWNTO 0);
ATTRIBUTE syn_encoding : STRING;
ATTRIBUTE syn_encoding OF current_state : SIGNAL IS "sequential";
```

By default, the state machine logic is optimized for speed and area, which may be potentially undesirable for critical systems. The safe value generates extra control logic to force the state machine to the reset state if an invalid state is reached.

1.9.4.1. FSM Explorer in Synplify Pro and Premier

The Synplify Pro and Premier software use the FSM Explorer to explore different encoding styles for a state machine automatically, and then implement the best encoding based on the overall design constraints. The FSM Explorer uses the FSM Compiler to identify and extract state machines from a design. However, unlike the FSM Compiler, which chooses the encoding style based on the number of states, the FSM Explorer attempts several different encoding styles before choosing a specific one. The trade-off is that the compilation requires more time to analyze the state machine, but finds an optimal encoding scheme for the state machine.

1.9.5. Optimization Attributes and Options

1.9.5.1. Retiming in Synplify Pro and Premier

The Synplify Pro and Premier software can retime a design, which can improve the timing performance of sequential circuits by moving registers (register balancing) across combinational elements. Be aware that retimed registers incur name changes. You can retime your design from **Implementation Options** or you can use the `syn_allow_retiming` attribute.

1.9.5.2. Maximum Fan-Out

When your design has critical path nets with high fan-out, use the `syn_maxfan` attribute to control the fan-out of the net. Setting this attribute for a specific net results in the replication of the driver of the net to reduce overall fan-out. The `syn_maxfan` attribute takes an integer value and applies it to inputs or registers. The `syn_maxfan` attribute cannot be used to duplicate control signals. The minimum allowed value of the attribute is 4. Using this attribute might result in increased logic resource utilization, thus straining routing resources, which can lead to long compilation times and difficult fitting.

If you must duplicate an output register or an output enable register, you can create a register for each output pin by using the `syn_useioff` attribute.

1.9.5.3. Preserving Nets

During synthesis, the compiler maintains ports, registers, and instantiated components. However, some nets cannot be maintained to create an optimized circuit. Applying the `syn_keep` directive overrides the optimization of the compiler and preserves the net during synthesis. The `syn_keep` directive is a Boolean data type value and can be applied to wires (Verilog HDL) and signals (VHDL). Setting the value to **true** preserves the net through synthesis.

1.9.5.4. Register Packing

Intel devices allow register packing into I/O cells. Intel recommends allowing the Intel Quartus Prime software to make the I/O register assignments. However, you can control register packing with the `syn_useioff` attribute. The `syn_useioff` attribute is a Boolean data type value that can be applied to ports or entire modules. Setting the value to **1** instructs the compiler to pack the register into an I/O cell. Setting the value to **0** prevents register packing in both the Synplify and Intel Quartus Prime software.

1.9.5.5. Resource Sharing

The Synplify software uses resource sharing techniques during synthesis, by default, to reduce area. Turning off the **Resource Sharing** option on the **Options** tab of the **Implementation Options** dialog box improves performance results for some designs. You can also turn off the option for a specific module with the `syn_sharing` attribute. If you turn off this option, be sure to check the results to verify improvement in timing performance. If there is no improvement, turn on **Resource Sharing**.

1.9.5.6. Preserving Hierarchy

The Synplify software performs cross-boundary optimization by default, which causes the design to flatten to allow optimization. You can use the `syn_hier` attribute to override the default compiler settings. The `syn_hier` attribute applies a string value to modules, architectures, or both. Setting the value to **hard** maintains the boundaries of a module, architecture, or both, but allows constant propagation. Setting the value to **locked** prevents all cross-boundary optimizations. Use the **locked** setting with the partition setting to create separate design blocks and multiple output netlists.

By default, the Synplify software generates a hierarchical `.vqm` file. To flatten the file, set the `syn_netlist_hierarchy` attribute to **0**.

1.9.5.7. Register Input and Output Delays

Two advanced options, `define_reg_input_delay` and `define_reg_output_delay`, can speed up paths feeding a register, or coming from a register, by a specific number of nanoseconds. The Synplify software attempts to meet the global clock frequency goals for a design as well as the individual clock frequency goals (set with the `define_clock` attribute). You can use these attributes to add a delay to paths feeding into or out of registers to further constrain critical paths. You can slow down a path that is too highly optimized by setting this attributes to a negative number.

The `define_reg_input_delay` and `define_reg_output_delay` options are useful to close timing if your design does not meet timing goals, because the routing delay after placement and routing exceeds the delay predicted by the Synplify software. Rerun synthesis using these options, specifying the actual routing delay (from place-and-route results) so that the tool can meet the required clock frequency. Synopsys recommends that for best results, do not make these assignments too aggressively. For example, you can increase the routing delay value, but do not also use the full routing delay from the last compilation.

In the SCOPE constraint window, the registers panel contains the following options:

- **Register**—Specifies the name of the register. If you have initialized a compiled design, select the name from the list.
- **Type**—Specifies whether the delay is an input or output delay.
- **Route**—Shrinks the effective period for the constrained registers by the specified value without affecting the clock period that is forward-annotated to the Intel Quartus Prime software.

Use the following Tcl command syntax to specify an input or output register delay in nanoseconds.

Example 3. Input and Output Register Delay

```
define_reg_input_delay {<register>} -route <delay in ns>
define_reg_output_delay {<register>} -route <delay in ns>
```

1.9.5.8. syn_direct_enable

This attribute controls the assignment of a clock-enable net to the dedicated enable pin of a register. With this attribute, you can direct the Synplify mapper to use a particular net as the only clock enable when the design has multiple clock enable candidates.

To use this attribute as a compiler directive to infer registers with clock enables, enter the `syn_direct_enable` directive in your source code, instead of the SCOPE spreadsheet.

The `syn_direct_enable` data type is Boolean. A value of **1** or **true** enables net assignment to the clock-enable pin. The following is the syntax for Verilog HDL:

```
object /* synthesis syn_direct_enable = 1 */ ;
```

1.9.5.9. I/O Standard

For certain Intel devices, specify the I/O standard type for an I/O pad in the design with the **I/O Standard** panel in the Synplify SCOPE window.

The Synplify SDC syntax for the `define_io_standard` constraint, in which the `delay_type` must be either `input_delay` or `output_delay`.

Example 4. define_io_standard Constraint

```
define_io_standard [-disable|-enable] {<objectName>} -delay_type \
[input_delay|output_delay] <columnTclName>{<value>} [<columnTclName>{<value>}...]
```

For details about supported I/O standards, refer to the *Synopsys FPGA Synthesis Reference Manual.*

1.9.6. Intel-Specific Attributes

You can use the `altera_chip_pin_lc`, `altera_io_powerup`, and `altera_io_opendrain` attributes with specific Intel device features, which are forward-annotated to the Intel Quartus Prime project, and are used during place-and-route.

1.9.6.1. altera_chip_pin_lc

Use the `altera_chip_pin_lc` attribute to make pin assignments. This attribute applies a string value to inputs and outputs. Use the attribute only on the ports of the top-level entity in the design. Do not use this attribute to assign pin locations from entities at lower levels of the design hierarchy.

Note: The `altera_chip_pin_lc` attribute is not supported for any MAX series device.

In the SCOPE window, set the value of the `altera_chip_pin_lc` attribute to a pin number or a list of pin numbers.

You can use VHDL code for making location assignments for supported Intel devices. Pin location assignments for these devices are written to the output `.tcl` file.

Note: The `data_out` signal is a 4-bit signal; `data_out[3]` is assigned to pin 14 and `data_out[0]` is assigned to pin 15.

Example 5. Making Location Assignments in VHDL

```
ENTITY sample (data_in : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
    data_out: OUT STD_LOGIC_VECTOR (3 DOWNTO 0));
ATTRIBUTE altera_chip_pin_lc : STRING;
ATTRIBUTE altera_chip_pin_lc OF data_out : SIGNAL IS "14, 5, 16, 15";
```

1.9.6.2. altera_io_powerup

Use the `altera_io_powerup` attribute to define the power-up value of an I/O register that has no set or reset. This attribute applies a string value (**high**|**low**) to ports with I/O registers. By default, the power-up value of the I/O register is set to **low**.

1.9.6.3. altera_io_opendrain

Use the `altera_io_opendrain` attribute to specify open-drain mode I/O ports. This attribute applies a boolean data type value to outputs or bidirectional ports for devices that support open-drain mode.

1.10. Guidelines for Intel FPGA IP Cores and Architecture-Specific Features

Intel provides parameterizable IP cores, including LPMs, device-specific Intel FPGA IP cores, and IP available through the Intel FPGA IP Partners Program (AMPP^{SM}). You can use IP cores by instantiating them in your HDL code, or by inferring certain IP cores from generic HDL code.

You can instantiate an IP core in your HDL code with the IP Catalog and configure the IP core with the Parameter Editor, or instantiate the IP core using the port and parameter definition. The IP Catalog and Parameter Editor provide a graphical interface within the Intel Quartus Prime software to customize any available Intel FPGA IP core for the design.

The Synplify software also automatically recognizes certain types of HDL code, and infers the appropriate Intel FPGA IP core when an IP core provides optimal results. The Synplify software provides options to control inference of certain types of IP cores.

Related Information

Hardware Description Language Support on page 6

1.10.1. Instantiating Intel FPGA IP Cores with the IP Catalog

When you use the IP Catalog and Parameter Editor to set up and configure an IP core, the IP Catalog creates a VHDL or Verilog HDL wrapper file <*output file*>`.v|vhd` that instantiates the IP core.

The Synplify software uses the Intel Quartus Prime timing and resource estimation netlist feature to report more accurate resource utilization and timing performance estimates, and uses timing-driven optimization, instead of treating the IP core as a "black box." Including the generated IP core variation wrapper file in your Synplify project, gives the Synplify software complete information about the IP core.

Note: There is an option in the Parameter Editor to generate a netlist for resource and timing estimation. This option is not recommended for the Synplify software because the software automatically generates this information in the background without a separate netlist. If you do create a separate netlist <*output file*>`_syn.v` and use that file in your synthesis project, you must also include the <*output file*>`.v|vhd` file in your Intel Quartus Prime project.

Verify that the correct Intel Quartus Prime version is specified in the Synplify software before compiling the generated file to ensure that the software uses the correct library definitions for the IP core. The **Quartus Version** setting must match the version of the Intel Quartus Prime software used to generate the customized IP core.

In addition, ensure that the *QUARTUS_ROOTDIR* environment variable specifies the installation directory location of the correct Intel Quartus Prime version. The Synplify software uses this information to launch the Intel Quartus Prime software in the background. The environment variable setting must match the version of the Intel Quartus Prime software used to generate the customized IP core.

Related Information

- Specifying the Intel Quartus Prime Software Version on page 6
- Using the Intel Quartus Prime Software to Run the Synplify Software on page 8

1.10.1.1. Instantiating Intel FPGA IP Cores with IP Catalog Generated Verilog HDL Files

If you turn on the <*output file*>`_inst.v` option on the Parameter Editor, the IP Catalog generates a Verilog HDL instantiation template file for use in your Synplify design. The instantiation template file, <*output file*>`_inst.v`, helps to instantiate the IP core variation wrapper file, <*output file*>`.v`, in your top-level design. Include the IP core variation wrapper file <*output file*>`.v` in your Synplify project. The Synplify software includes the IP core information in the output `.vqm` netlist file. You do not need to include the generated IP core variation wrapper file in your Intel Quartus Prime project.

1.10.1.2. Instantiating Intel FPGA IP Cores with IP Catalog Generated VHDL Files

If you turn on the <*output file*>`.cmp` and <*output file*>`_inst.vhd` options on the parameter editor, the IP Catalog generates a VHDL component declaration file and a VHDL instantiation template file for use in your Synplify design. These files can help you instantiate the IP core variation wrapper file, <*output file*>`.vhd`, in your top-level design. Include the <*output file*>`.vhd` in your Synplify project. The Synplify software includes the IP core information in the output `.vqm` netlist file. You do not need to include the generated IP core variation wrapper file in your Intel Quartus Prime project.

1.10.1.3. Changing Synplify’s Default Behavior for Instantiated Intel FPGA IP Cores

By default, the Synplify software automatically opens the Intel Quartus Prime software in the background to generate a resource and timing estimation netlist for IP cores.

You might want to change this behavior to reduce run times in the Synplify software, because generating the netlist files can take several minutes for large designs, or if the Synplify software cannot access your Intel Quartus Prime software installation to generate the files. Changing this behavior might speed up the compilation time in the Synplify software, but the Quality of Results (QoR) might be reduced.

The Synplify software directs the Intel Quartus Prime software to generate information in two ways:

- Some IP cores provide a “clear box” model—the Synplify software fully synthesizes this model and includes the device architecture-specific primitives in the output `.vqm` netlist file.
- Other IP cores provide a “gray box” model—the Synplify software reads the resource information, but the netlist does not contain all the logic functionality.

  Note: You need to turn on **Generate netlist** when using the gray box model. For more information, see the Intel Quartus Prime online help.

For these IP cores, the Synplify software uses the logic information for resource and timing estimation and optimization, and then instantiates the IP core in the output `.vqm` netlist file so the Intel Quartus Prime software can implement the appropriate device primitives. By default, the Synplify software uses the clear box model when available, and otherwise uses the gray box model.

Related Information

- Including Files for Intel Quartus Prime Placement and Routing Only on page 23
- Synplify Synthesis Techniques with the Intel Quartus Prime Software online training
  Includes more information about design flows using clear box model and gray box model.

1.10.1.4. Instantiating Intellectual Property with the IP Catalog and Parameter Editor

Many Intel FPGA IP cores include a resource and timing estimation netlist that the Synplify software uses to report more accurate resource utilization and timing performance estimates, and uses timing-driven optimization rather than a black box function.

To create this netlist file, perform the following steps:

1. Select the IP core in the IP Catalog.
2. Click **Next** to open the Parameter Editor.
3. Click **Set Up Simulation**, which sets up all the EDA options.
4. Turn on the **Generate netlist** option to generate a netlist for resource and timing estimation and click **OK**.
5. Click **Generate** to generate the netlist file.

The Intel Quartus Prime software generates a file <*output file*>`_syn.v`. This netlist contains the gray box information for resource and timing estimation, but does not contain the actual implementation. Include this netlist file in your Synplify project. Next, include the IP core variation wrapper file <*output file*>`.v|vhd` in the Intel Quartus Prime project along with your Synplify `.vqm` output netlist.

If your IP core does not include a resource and timing estimation netlist, the Synplify software must treat the IP core as a black box.

Related Information

Including Files for Intel Quartus Prime Placement and Routing Only on page 23

1.10.1.5. Instantiating Black Box IP Cores with Generated Verilog HDL Files

Use the `syn_black_box` compiler directive to declare a module as a black box. The top-level design files must contain the IP port-mapping and a hollow-body module declaration. Apply the `syn_black_box` directive to the module declaration in the top-level file or a separate file included in the project so that the Synplify software recognizes the module is a black box. The software compiles successfully without this directive, but reports an additional warning message. Using this directive allows you to add other directives.

The example shows a top-level file that instantiates `my_verilogIP.v`, which is a simple customized variation generated by the IP Catalog.

Example 6. Sample Top-Level Verilog HDL Code with Black Box Instantiation of IP

```
module top (clk, count);
    input clk;
    output [7:0] count;
    my_verilogIP verilogIP_inst (.clock (clk), .q (count));
endmodule
// Module declaration
// The following attribute is added to create a
// black box for this module.
module my_verilogIP (clock, q) /* synthesis syn_black_box */;
    input clock;
    output [7:0] q;
endmodule
```

1.10.1.6. Instantiating Black Box IP Cores with Generated VHDL Files

Use the `syn_black_box` compiler directive to declare a component as a black box. The top-level design files must contain the IP core variation component declaration and port-mapping. Apply the `syn_black_box` directive to the component declaration in the top-level file. The software compiles successfully without this directive, but reports an additional warning message. Using this directive allows you to add other directives.

The example shows a top-level file that instantiates `my_vhdlIP.vhd`, which is a simplified customized variation generated by the IP Catalog.

Example 7. Sample Top-Level VHDL Code with Black Box Instantiation of IP

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

ENTITY top IS
    PORT (
        clk: IN STD_LOGIC ;
        count: OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
    );
END top;

ARCHITECTURE rtl OF top IS
COMPONENT my_vhdlIP
    PORT (
        clock: IN STD_LOGIC ;
        q: OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
    );
end COMPONENT;
attribute syn_black_box : boolean;
attribute syn_black_box of my_vhdlIP: component is true;
BEGIN
    vhdlIP_inst : my_vhdlIP PORT MAP (
        clock => clk,
        q => count
    );
END rtl;
```

1.10.1.7. Other Synplify Software Attributes for Creating Black Boxes

Instantiating IP as a black box does not provide visibility into the IP for the synthesis tool. Thus, it does not take full advantage of the synthesis tool's timing-driven optimization. For better timing optimization, especially if the black box does not have registered inputs and outputs, add timing models to black boxes by adding the `syn_tpd`, `syn_tsu`, and `syn_tco` attributes.

Example 8. Adding Timing Models to Black Boxes in Verilog HDL

```
module ram32x4(z,d,addr,we,clk);
    /* synthesis syn_black_box syn_tcol="clk->z[3:0]=4.0"
        syn_tpd1="addr[3:0]->[3:0]=8.0"
        syn_tsu1="addr[3:0]->clk=2.0"
        syn_tsu2="we->clk=3.0" */
    output [3:0]z;
    input[3:0]d;
    input[3:0]addr;
    input we
    input clk
endmodule
```

The following additional attributes are supported by the Synplify software to communicate details about the characteristics of the black box module within the HDL code:

- `syn_resources`—Specifies the resources used in a particular black box.
- `black_box_pad_pin`—Prevents mapping to I/O cells.
- `black_box_tri_pin`—Indicates a tri-stated signal.

For more information about applying these attributes, refer to the *Synopsys FPGA Synthesis Reference Manual*.

1.10.2. Including Files for Intel Quartus Prime Placement and Routing Only

In the Synplify software, you can add files to your project that are used only during placement and routing in the Intel Quartus Prime software. This can be useful if you have gray or black boxes for Synplify synthesis that require the full design files to be compiled in the Intel Quartus Prime software.

You can also set the option in a script using the `-job_owner par` option.

The example shows how to define files for a Synplify project that includes a top-level design file, a gray box netlist file, an IP wrapper file, and an encrypted IP file. With these files, the Synplify software writes an empty instantiation of “core” in the `.vqm` file and uses the gray box netlist for resource and timing estimation. The files `core.v` and `core_enc8b10b.v` are not compiled by the Synplify software, but are copied into the place-and-route directory. The Intel Quartus Prime software compiles these files to implement the “core” IP block.

Example 9. Commands to Define Files for a Synplify Project

```
add_file -verilog -job_owner par "core_enc8b10b.v"
add_file -verilog -job_owner par "core.v"
add_file -verilog "core_gb.v"
add_file -verilog "top.v"
```

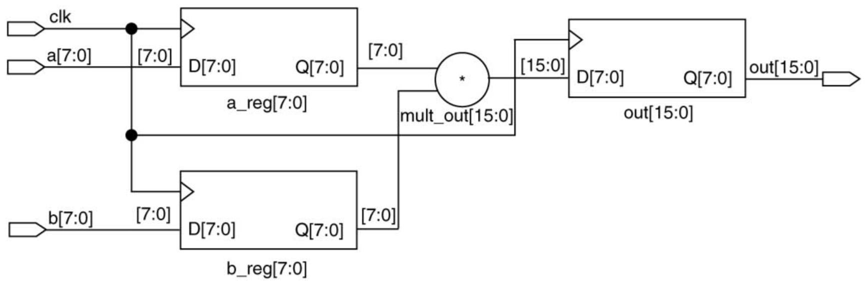
1.10.3. Inferring Intel FPGA IP Cores from HDL Code

The Synplify software uses Behavior Extraction Synthesis Technology (BEST) algorithms to infer high-level structures such as RAMs, ROMs, operators, FSMs, and DSP multiplication operations. Then, the Synplify software keeps the structures abstract for as long as possible in the synthesis process. This allows the use of technology-specific resources to implement these structures by inferring the appropriate Intel FPGA IP core when an IP core provides optimal results.

1.10.3.1. Inferring Multipliers

The figure shows the HDL Analyst view of an unsigned 8 × 8 multiplier with two pipeline stages after synthesis in the Synplify software. This multiplier is converted into an ALTMULT_ADD or ALTMULT_ACCUM IP core. For devices with DSP blocks, the software might implement the function in a DSP block instead of regular logic, depending on device utilization. For some devices, the software maps directly to DSP block device primitives instead of instantiating an IP core in the **.vqm** file.

Figure 2. HDL Analyst View of LPM_MULT IP Core (Unsigned 8x8 Multiplier with Pipeline=2)



1.10.3.1.1. Resource Balancing

While mapping multipliers to DSP blocks, the Synplify software performs resource balancing for optimum performance.

Intel devices have a fixed number of DSP blocks, which includes a fixed number of embedded multipliers. If the design uses more multipliers than are available, the Synplify software automatically maps the extra multipliers to logic elements (LEs), or adaptive logic modules (ALMs).

If a design uses more multipliers than are available in the DSP blocks, the Synplify software maps the multipliers in the critical paths to DSP blocks. Next, any wide multipliers, which might or might not be in the critical paths, are mapped to DSP blocks. Smaller multipliers and multipliers that are not in the critical paths might then be implemented in the logic (LEs or ALMs). This ensures that the design fits successfully in the device.

1.10.3.1.2. Controlling the DSP Block Inference

You can implement multipliers in DSP blocks or in logic in Intel devices that contain DSP blocks. You can control this implementation through attribute settings in the Synplify software.

1.10.3.1.3. Signal Level Attribute

You can control the implementation of individual multipliers by using the `syn_multstyle` attribute as shown in the following Verilog HDL code (where *<signal_name>* is the name of the signal):

```
<signal_name> /* synthesis syn_multstyle = "logic" */;
```

The `syn_multstyle` attribute applies to wires only; it cannot be applied to registers.

Table 3. DSP Block Attribute Setting in the Synplify Software

Attribute Name	Value	Description
syn_multstyle	lpm_mult	LPM function inferred and multipliers implemented in DSP blocks.
	logic	LPM function not inferred and multipliers implemented as LEs by the Synplify software.
	block_mult	DSP IP core is inferred and multipliers are mapped directly to DSP block device primitives (for supported devices).

Example 10. Signal Attributes for Controlling DSP Block Inference in Verilog HDL Code

```
module mult(a,b,c,r,en);
    input [7:0] a,b;
    output [15:0] r;
    input [15:0] c;
    input en;
    wire [15:0] temp /* synthesis syn_multstyle="logic" */;

    assign temp = a*b;
    assign r = en ? temp : c;
endmodule
```

Example 11. Signal Attributes for Controlling DSP Block Inference in VHDL Code

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

entity onereg is port (
    r : out std_logic_vector (15 downto 0);
    en : in std_logic;
    a : in std_logic_vector (7 downto 0);
    b : in std_logic_vector (7 downto 0);
    c : in std_logic_vector (15 downto 0);
    );
end onereg;

architecture beh of onereg is
signal temp : std_logic_vector (15 downto 0);
attribute syn_multstyle : string;
attribute syn_multstyle of temp : signal is "logic";

begin
    temp <= a * b;
    r <= temp when en='1' else c;
end beh;
```

1.10.3.2. Inferring RAM

When a RAM block is inferred from an HDL design, the Synplify software uses an Intel FPGA IP core to target the device memory architecture. For some devices, the Synplify software maps directly to memory block device primitives instead of instantiating an IP core in the `.vqm` file.

Follow these guidelines for the Synplify software to successfully infer RAM in a design:

- The address line must be at least two bits wide.
- Resets on the memory are not supported. Refer to the device family documentation for information about whether read and write ports must be synchronous.
- Some Verilog HDL statements with blocking assignments might not be mapped to RAM blocks, so avoid blocking statements when modeling RAMs in Verilog HDL.

For some device families, the `syn_ramstyle` attribute specifies the implementation to use for an inferred RAM. You can apply the `syn_ramstyle` attribute globally to a module or a RAM instance, to specify `registers` or `block_ram` values. To turn off RAM inference, set the attribute value to `registers`.

When inferring RAM for some Intel device families, the Synplify software generates additional bypass logic. This logic is generated to resolve a half-cycle read/write behavior difference between the RTL and post-synthesis simulations. The RTL simulation shows the memory being updated on the positive edge of the clock; the post-synthesis simulation shows the memory being updated on the negative edge of the clock. To eliminate bypass logic, the output of the RAM must be registered. By adding this register, the output of the RAM is seen after a full clock cycle, by which time the update has occurred, thus eliminating the need for bypass logic.

For devices with TriMatrix memory blocks, disable the creation of glue logic by setting the `syn_ramstyle` value to `no_rw_check`. Set `syn_ramstyle` to `no_rw_check` to disable the creation of glue logic in dual-port mode.

Example 12. VHDL Code for Inferred Dual-Port RAM

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

ENTITY dualport_ram IS
PORT ( data_out: OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
    data_in: IN STD_LOGIC_VECTOR (7 DOWNTO 0)
    wr_addr, rd_addr: IN STD_LOGIC_VECTOR (6 DOWNTO 0);
    we: IN STD_LOGIC);
    clk: IN STD_LOGIC);
END dualport_ram;

ARCHITECTURE ram_infer OF dualport_ram IS
TYPE Mem_Type IS ARRAY (127 DOWNTO 0) OF STD_LOGIC_VECOR (7 DOWNTO 0);
SIGNAL mem; Mem_Type;
SIGNAL addr_reg: STD_LOGIC_VECTOR (6 DOWNTO 0);

BEGIN
    data_out <= mem (CONV_INTEGER(rd_addr));
    PROCESS (clk, we, data_in) BEGIN
        IF (clk='1' AND clk'EVENT) THEN
            IF (we='1') THEN
                mem(CONV_INTEGER(wr_addr)) <= data_in;
            END IF;
        END IF;
    END PROCESS;
END ram_infer;
```

Example 13. VHDL Code for Inferred Dual-Port RAM Preventing Bypass Logic

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

ENTITY dualport_ram IS
PORT ( data_out: OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
    data_in : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
    wr_addr, rd_addr : IN STD_LOGIC_VECTOR (6 DOWNTO 0);
    we : IN STD_LOGIC;
    clk : IN STD_LOGIC);
END dualport_ram;

ARCHITECTURE ram_infer OF dualport_ram IS
TYPE Mem_Type IS ARRAY (127 DOWNTO 0) OF STD_LOGIC_VECTOR (7 DOWNTO 0);
SIGNAL mem : Mem_Type;
SIGNAL addr_reg : STD_LOGIC_VECTOR (6 DOWNTO 0);
SIGNAL tmp_out : STD_LOGIC_VECTOR (7 DOWNTO 0); --output register

BEGIN
    tmp_out <= mem (CONV_INTEGER (rd_addr));
    PROCESS (clk, we, data_in) BEGIN
        IF (clk='1' AND clk'EVENT) THEN
            IF (we='1') THEN
                mem(CONV_INTEGER(wr_addr)) <= data_in;
            END IF;
            data_out <= tmp_out; --registers output preventing
                                 -- bypass logic generation
        END IF;
    END PROCESS;
END ram_infer;
```

1.10.3.3. RAM Initialization

Use the Verilog HDL `$readmemb` or `$readmemh` system tasks in your HDL code to initialize RAM memories. The Synplify compiler forward-annotates the initialization values in the `.srs` (technology-independent RTL netlist) file and the mapper generates the corresponding hexadecimal memory initialization (`.hex`) file. One `.hex` file is created for each of the `altsyncram` IP cores that are inferred in the design. The `.hex` file is associated with the `altsyncram` instance in the `.vqm` file using the `init_file` attribute.

The examples show how RAM can be initialized through HDL code, and how the corresponding `.hex` file is generated using Verilog HDL.

Example 14. Using $readmemb System Task to Initialize an Inferred RAM in Verilog HDL Code

```
initial
begin
    $readmemb("mem.ini", mem);
end
always @(posedge clk)
begin
    raddr_reg <= raddr;
    if(we)
        mem[waddr] <= data;
end
```

Example 15. Sample of .vqm Instance Containing Memory Initialization File

```
altsyncram mem_hex( .wren_a(we),.wren_b(GND),...);

defparam mem_hex.lpm_type = "altsyncram";
defparam mem_hex.operation_mode = "Dual_Port";
...
defparam mem_hex.init_file = "mem_hex.hex";
```

1.10.3.4. Inferring ROM

When a ROM block is inferred from an HDL design, the Synplify software uses an Intel FPGA IP core to target the device memory architecture. For some devices, the Synplify software maps directly to memory block device atoms instead of instantiating an IP core in the `.vqm` file.

Follow these guidelines for the Synplify software to successfully infer ROM in a design:

- The address line must be at least two bits wide.
- The ROM must be at least half full.
- A CASE or IF statement must make 16 or more assignments using constant values of the same width.

1.10.3.5. Inferring Shift Registers

The Synplify software infers shift registers for sequential shift components so that they can be placed in dedicated memory blocks in supported device architectures using the ALTSHIFT_TAPS IP core.

If necessary, set the implementation style with the `syn_srlstyle` attribute. If you do not want the components automatically mapped to shift registers, set the value to `registers`. You can set the value globally, or on individual modules or registers.

For some designs, turning off shift register inference improves the design performance.

1.11. Incremental Compilation and Block-Based Design

As designs become more complex and designers work in teams, a block-based incremental design flow is often an effective design approach. In an incremental compilation flow, you can make changes to part of the design while maintaining the placement and performance of unchanged parts of the design. Design iterations are made dramatically faster by focusing new compilations on particular design partitions and merging results with previous compilation results of other partitions. You can perform optimization on individual subblocks and then preserve the results before you integrate the blocks into a final design and optimize it at the top-level.

MultiPoint synthesis, which is available for certain device technologies in the Synplify Pro and Premier software, provides an automated block-based incremental synthesis flow. The MultiPoint feature manages a design hierarchy to let you design incrementally and synthesize designs that take too long for synthesis of the entire project. MultiPoint synthesis allows different netlist files to be created for different sections of a design hierarchy and supports the Intel Quartus Prime incremental compilation methodology. This feature also ensures that only those sections of a design that have been updated are resynthesized when the design is compiled, reducing synthesis run time and preserving the results for the unchanged blocks. You can change and resynthesize one section of a design without affecting other sections.

You can also partition your design and create different netlist files manually with the Synplify software by creating a separate project for the logic in each partition of the design. Creating different netlist files for each partition of the design also means that each partition can be independent of the others.

Hierarchical design methodologies can improve the efficiency of your design process, providing better design reuse opportunities and fewer integration problems when working in a team environment. When you use these incremental synthesis methodologies, you can take advantage of incremental compilation in the Intel Quartus Prime software. You can perform placement and routing on only the changed partitions of the design, which reduces place-and-route time and preserves your fitting results.

1.11.1. Design Flow for Incremental Compilation

The following steps describe the general incremental compilation flow when using these features of the Intel Quartus Prime software:

1. Create Verilog HDL or VHDL design files.
2. Determine which hierarchical blocks you want to treat as separate partitions in your design.
3. Set up your design using the MultiPoint synthesis feature or separate projects so that a separate netlist file is created for each design partition.
4. If using separate projects, disable I/O pad insertion in the implementations for lower-level partitions.
5. Compile and map each partition in the Synplify software, making constraints as you would in a non-incremental design flow.
6. Import the **.vqm** netlist and **.tcl** file for each partition into the Intel Quartus Prime software and set up the Intel Quartus Prime project(s) for incremental compilation.
7. Compile your design in the Intel Quartus Prime software and preserve the compilation results with the post-fit netlist in incremental compilation.
8. When you make design or synthesis optimization changes to part of your design, resynthesize only the partition you modified to generate a new netlist and **.tcl** file. Do not regenerate netlist files for the unmodified partitions.
9. Import the new netlist and **.tcl** file into the Intel Quartus Prime software and recompile the design in the Intel Quartus Prime software with incremental compilation.

1.11.2. Creating a Design with Separate Netlist Files for Incremental Compilation

The first stage of a hierarchical or incremental design flow is to ensure that different parts of your design do not affect each other. Ensure that you have separate netlists for each partition in your design so you can take advantage of incremental compilation in the Intel Quartus Prime software. If the entire design is in one netlist file, changes in one partition might affect other partitions because of possible node name changes when you resynthesize the design.

To ensure proper functionality of the synthesis flow, create separate netlist files only for modules and entities. In addition, each module or entity requires its own design file. If two different modules are in the same design file, but are defined as being part of different partitions, incremental compilation cannot be maintained since both partitions must be recompiled when one module is changed.

Intel recommends that you register all inputs and outputs of each partition. This makes logic synchronous, and avoids any delay penalty on signals that cross partition boundaries.

If you use boundary tri-states in a lower-level block, the Synplify software pushes, or bubbles, the tri-states through the hierarchy to the top-level to use the tri-state drivers on output pins of Intel devices. Because bubbling tri-states requires optimizing through hierarchies, lower-level tri-states are not supported with a block-based compilation methodology. Use tri-state drivers only at the external output pins of the device and in the top-level block in the hierarchy.

You can generate multiple **.vqm** netlist files with the MultiPoint synthesis flow in the Synplify Pro and Premier software, or by manually creating separate Synplify projects and creating a black box for each block that you want to designate as a separate design partition.

In the MultiPoint synthesis flow in the Synplify Pro and Premier software, you create multiple **.vqm** netlist files from one easy-to-manage, top-level synthesis project. By using the manual black box method, you have multiple synthesis projects, which might be required for certain team-based or bottom-up designs where a single top-level project is not desired.

After you have created multiple **.vqm** files using one of these two methods, you must create the appropriate Intel Quartus Prime projects to place-and-route the design.

1.11.3. Using MultiPoint Synthesis with Incremental Compilation

This topic describes how to generate multiple **.vqm** files using the Synplify Pro and Premier software MultiPoint synthesis flow. You must first set up your constraint file and Synplify options, then apply the appropriate Compile Point settings to write multiple **.vqm** files and create design partition assignments for incremental compilation.

1.11.3.1. Set Compile Points and Create Constraint Files

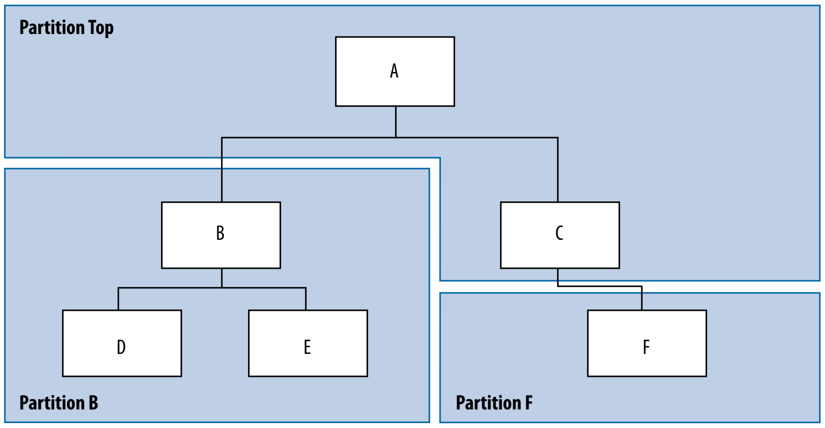
The MultiPoint flow lets you segment a design into smaller synthesis units, called Compile Points. The synthesis software treats each Compile Point as a partition for incremental mapping, which allows you to isolate and work on each Compile Point module as independent segments of the larger design without impacting other design modules. A design can have any number of Compile Points, and Compile Points can be nested. The top-level module is always treated as a Compile Point.

Compile Points are optimized in isolation from their parent, which can be another Compile Point or a top-level design. Each block created with a Compile Point is unaffected by critical paths or constraints on its parent or other blocks. A Compile Point is independent, with its own individual constraints. During synthesis, any Compile Points that have not yet been synthesized are synthesized before the top level. Nested Compile Points are synthesized before the parent Compile Points in which they are contained. When you apply the appropriate setting for the Compile Point, a separate netlist is created for that Compile Point, isolating that logic from any other logic in the design.

The figure shows an example of a design hierarchy that is split into multiple partitions. The top-level block of each partition can be synthesized as a separate Compile Point.

Send Feedback

Figure 3. Partitions in a Hierarchical Design



In this case, modules A, B, and F are Compile Points. The top-level Compile Point consists of the top-level block in the design (that is, block A in this example), including the logic that is not defined under another Compile Point. In this example, the design for top-level Compile Point A also includes the logic in one of its subblocks, C. Because block F is defined as its own Compile Point, it is not treated as part of the top-level Compile Point A. Another separate Compile Point B contains the logic in blocks B, D, and E. One netlist is created for the top-level module A and submodule C, another netlist is created for B and its submodules D and E, while a third netlist is created for F.

Apply Compile Points to the module, or to the architecture in the Synplify Pro SCOPE spreadsheet, or to the **.sdc** file. You cannot set a Compile Point in the Verilog HDL or VHDL source code. You can set the constraints manually using Tcl, by editing the **.sdc** file, or you can use the GUI.

1.11.3.1.1. Defining Compile Points With .tcl or .sdc Files

To set Compile Points with a **.tcl** or **.sdc** file, use the `define_compile_point` command.

Example 16. The define_compile_point Command

```
define_compile_point [-disable] {<objname>} -type {locked, partition}
```

<objname> represents any module in the design. The Compile Point type `{locked, partition}` indicates that the Compile Point represents a partition for the Intel Quartus Prime incremental compilation flow.

Each Compile Point has a set of constraint files that begin with the `define_current_design` command to set up the SCOPE environment, as follows:

```
define_current_design {<my_module>}
```

1.11.3.2. Additional Considerations for Compile Points

To ensure that changes to a Compile Point do not affect the top-level parent module, turn off the **Update Compile Point Timing Data** option in the **Implementation Options** dialog box. If this option is turned on, updates to a child module can impact the top-level module.

You can apply the `syn_allowed_resources` attribute to any Compile Point view to restrict the number of resources for a particular module.

When using Compile Points with incremental compilation, be aware of the following restrictions:

- To use Compile Points effectively, you must provide timing constraints (timing budgeting) for each Compile Point; the more accurate the constraints, the better your results are. Constraints are not automatically budgeted, so manual time budgeting is essential. Intel recommends that you register all inputs and outputs of each partition. This avoids any logic delay penalty on signals that cross-partition boundaries.
- When using the Synplify attribute `syn_useioff` to pack registers in the I/O Elements (IOEs) of Intel devices, these registers must be in the top-level module. Otherwise, you must direct the Intel Quartus Prime software to perform I/O register packing instead of the `syn_useioff` attribute. You can use the **Fast Input Register** or **Fast Output Register** options, or set I/O timing constraints and turn on **Optimize I/O cell register placement for timing** on the **Advanced Settings (Fitter)** dialog box in the Intel Quartus Prime software.
- There is no incremental synthesis support for top-level logic; any logic in the top-level is resynthesized during every compilation in the Synplify software.

For more information about using Compile Points and setting Synplify attributes and constraints for both top-level and lower-level Compile Points, refer to the *Synopsys FPGA Synthesis User Guide* and the *Synopsys FPGA Synthesis Reference Manual*.

1.11.3.3. Creating a Intel Quartus Prime Project for Compile Points and Multiple .vqm Files

During compilation, the Synplify Pro and Premier software creates a *<top-level project>***.tcl** file that provides the Intel Quartus Prime software with the appropriate constraints and design partition assignments, creating a partition for each **.vqm** file along with the information to set up a Intel Quartus Prime project.

Depending on your design methodology, you can create one Intel Quartus Prime project for all netlists or a separate Intel Quartus Prime project for each netlist. In the standard incremental compilation design flow, you create design partition assignments and optional LogicLock™ floorplan location assignments for each partition in the design within a single Intel Quartus Prime project. This methodology allows for the best quality of results and performance preservation during incremental changes to your design.

You might require a bottom-up design flow if each partition must be optimized separately, such as for third-party IP delivery. If you use this flow, Intel recommends you create a design floorplan to avoid placement conflicts between each partition. To follow this design flow in the Intel Quartus Prime software, create separate Intel Quartus Prime projects, export each design partition and incorporate them into a top-level design using the incremental compilation features to maintain placement results.

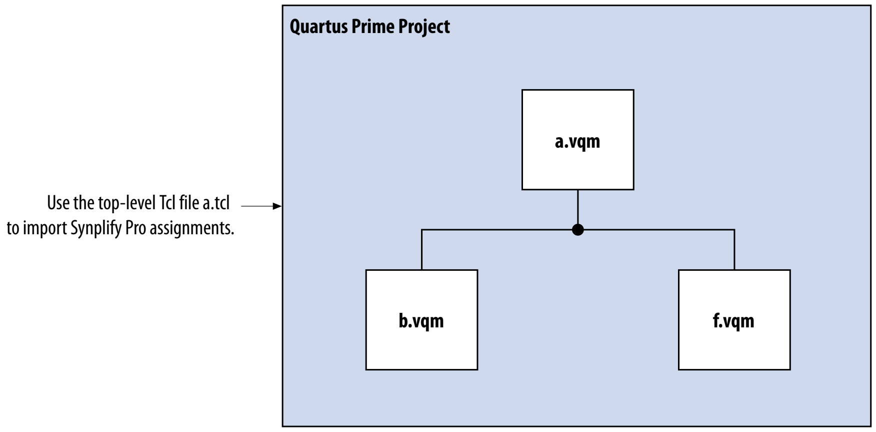
Related Information

Running the Intel Quartus Prime Software Manually With the Synplify-Generated Tcl Script on page 10

1.11.3.3.1. Creating a Single Intel Quartus Prime Project for a Standard Incremental Compilation Flow

Use the <*top-level project*>**.tcl** file that contains the Synplify assignments for all partitions within the project. This method allows you to import all the partitions into one Intel Quartus Prime project and optimize all modules within the project at once, while taking advantage of the performance preservation and compilation-time reduction that incremental compilation offers.

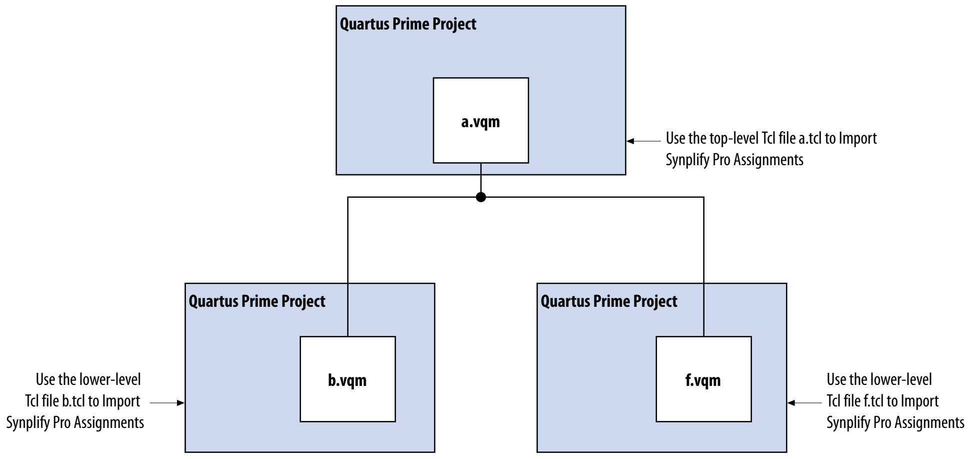
Figure 4. Design Flow Using Multiple .vqm Files with One Intel Quartus Prime Project


Quartus Prime Project
a.vqm
b.vqm
f.vqm
Use the top-level Tcl file a.tcl to import Synplify Pro assignments.

1.11.3.3.2. Creating Multiple Intel Quartus Prime Projects for a Bottom-Up Incremental Compilation Flow

Use the <*lower-level compile point*>**.tcl** files that contain the Synplify assignments for each Compile Point. Generate multiple Intel Quartus Prime projects, one for each partition and netlist in the design. The designers in the project can optimize their own partitions separately within the Intel Quartus Prime software and export the results for their own partitions. You can export the optimized subdesigns and then import them into one top-level Intel Quartus Prime project using incremental compilation to complete the design.

Figure 5. Design Flow Using Multiple .vqm Files with Multiple Intel Quartus Prime Projects


Quartus Prime Project
a.vqm
Use the top-level Tcl file a.tcl to Import Synplify Pro Assignments
Quartus Prime Project
b.vqm
Use the lower-level Tcl file b.tcl to Import Synplify Pro Assignments
Quartus Prime Project
f.vqm
Use the lower-level Tcl file f.tcl to Import Synplify Pro Assignments

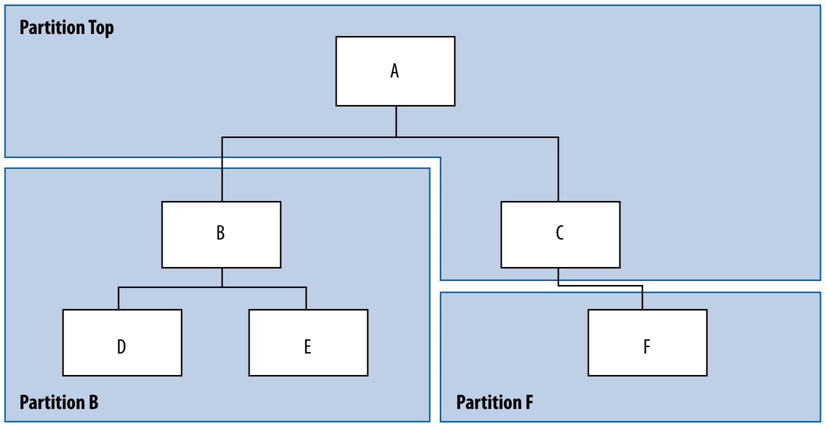
1.11.4. Creating Multiple .vqm Files for a Incremental Compilation Flow With Separate Synplify Projects

You can manually generate multiple **.vqm** files for a incremental compilation flow with black boxes and separate Synplify projects for each design partition. This manual flow is supported by versions of the Synplify software without the MultiPoint Synthesis feature.

1.11.4.1. Manually Creating Multiple .vqm Files With Black Boxes

To create multiple **.vqm** files manually in the Synplify software, create a separate project for each lower-level module and top-level design that you want to maintain as a separate **.vqm** file for an incremental compilation partition. Implement black box instantiations of lower-level partitions in your top-level project.

Figure 6. Partitions in a Hierarchical Design



The partition top contains the top-level block in the design (block A) and the logic that is not defined as part of another partition. In this example, the partition for top-level block A also includes the logic in one of its sub-blocks, block C. Because block F is contained in its own partition, it is not treated as part of the top-level partition A. Another separate partition, partition B, contains the logic in blocks B, D, and E. In a team-based design, engineers can work independently on the logic in different partitions. One netlist is created for the top-level module A and its submodule C, another netlist is created for module B and its submodules D and E, while a third netlist is created for module F.

1.11.4.1.1. Creating Multiple .vqm Files for this Design

To create multiple **.vqm** files for this design, follow these steps:

1. Generate a **.vqm** file for module B. Use **B.v/.vhd**, **D.v/.vhd**, and **E.v/.vhd** as the source files.
2. Generate a **.vqm** file for module F. Use **F.v/.vhd** as the source files.
3. Generate a top-level **.vqm** file for module A. Use **A.v/.vhd** and **C.v/.vhd** as the source files. Ensure that you use black box modules B and F, which were optimized separately in the previous steps.

Send Feedback

1.11.4.1.2. Creating Black Boxes in Verilog HDL

Any design block that is not defined in the project, or included in the list of files to be read for a project, is treated as a black box by the software. Use the `syn_black_box` attribute to indicate that you intend to create a black box for the module. In Verilog HDL, you must provide an empty module declaration for a module that is treated as a black box.

The example shows the **A.v** top-level file. Follow the same procedure for lower-level files that also contain a black box for any module beneath the current level hierarchy.

Example 17. Verilog HDL Black Box for Top-Level File A.v

```
module A (data_in, clk, e, ld, data_out);
    input data_in, clk, e, ld;
    output [15:0] data_out;

    wire [15:0] cnt_out;

    B U1 (.data_in (data_in),.clk(clk), .ld (ld),.data_out(cnt_out));
    F U2 (.d(cnt_out), .clk(clk), .e(e), .q(data_out));

    // Any other code in A.v goes here.
endmodule

// Empty Module Declarations of Sub-Blocks B and F follow here.
// These module declarations (including ports) are required for black boxes.

module B (data_in, clk, ld, data_out) /* synthesis syn_black_box */ ;
    input data_in, clk, ld;
    output [15:0} data_out;
endmodule

module F (d, clk, e, q) /* synthesis syn_black_box */ ;
    input [15:0] d;
    input clk, e;
    output [15:0] q;
endmodule
```

1.11.4.1.3. Creating Black Boxes in VHDL

Any design that is not defined in the project, or included in the list of files to be read for a project, is treated as a black box by the software. Use the `syn_black_box` attribute to indicate that you intend to treat the component as a black box. In VHDL, you must have a component declaration for the black box.

Although VHDL is not case-sensitive, a **.vqm** (a subset of Verilog HDL) file is case-sensitive. Entity names and their port declarations are forwarded to the **.vqm** file. Black box names and port declarations are also passed to the **.vqm** file. To prevent case-based mismatches, use the same capitalization for black box and entity declarations in VHDL designs.

The example shows the **A.vhd** top-level file. Follow this same procedure for any lower-level files that contain a black box for any block beneath the current level of hierarchy.

Example 18. VHDL Black Box for Top-Level File A.vhd

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

ENTITY A IS
PORT (data_in : IN INTEGER RANGE 0 TO 15;
    clk, e, ld : IN STD_LOGIC;
    data_out : OUT INTEGER RANGE 0 TO 15 );
```

```
END A;

ARCHITECTURE a_arch OF A IS

COMPONENT B PORT(
    data_in : IN INTEGER RANGE 0 TO 15;
    clk, ld : IN STD_LOGIC;
    d_out : OUT INTEGER RANGE 0 TO 15);
END COMPONENT;

COMPONENT F PORT(
    d : IN INTEGER RANGE 0 TO 15;
    clk, e: IN STD_LOGIC;
    q : OUT INTEGER RANGE 0 TO 15);
END COMPONENT;

attribute syn_black_box of B: component is true;
atrribute syn_black_box of F: component is true;

-- Other component declarations in A.vhd go here
signal cnt_out : INTEGER RANGE 0 TO 15;

BEGIN

U1 : B
PORT MAP (
    data_in => data_in,
    clk => clk,
    ld => ld,
    d_out => cnt_out );

U2 : F
PORT MAP (
    d => cnt_out,
    clk => clk,
    e => e,
    q => data_out );

-- Any other code in A.vhd goes here

END a_arch;
```

After you complete the steps above, you have a netlist for each partition of the design. These files are ready for use with the incremental compilation flow in the Intel Quartus Prime software.

1.11.4.2. Creating a Intel Quartus Prime Project for Multiple .vqm Files

The Synplify software creates a **.tcl** file for each **.vqm** file that provides the Intel Quartus Prime software with the appropriate constraints and information to set up a project.

Depending on your design methodology, you can create one Intel Quartus Prime project for all netlists or a separate Intel Quartus Prime project for each netlist. In the standard incremental compilation design flow, you create design partition assignments and optional LogicLock floorplan location assignments for each partition in the design within a single Intel Quartus Prime project. This methodology allows for the best quality of results and performance preservation during incremental changes to your design. You might require a bottom-up design flow where each partition must be optimized separately, such as for third-party IP delivery.

To perform this design flow in the Intel Quartus Prime software, create separate Intel Quartus Prime projects, export each design partition and incorporate it into a top-level design using the incremental compilation features to maintain the results.

Related Information

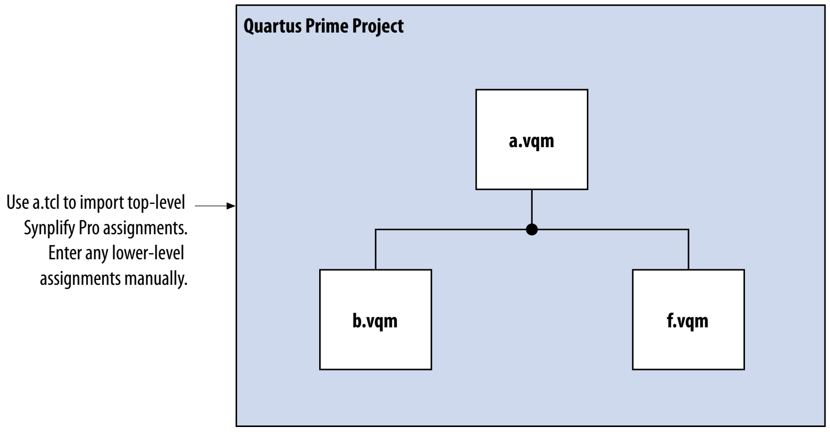
Running the Intel Quartus Prime Software Manually With the Synplify-Generated Tcl Script on page 10

1.11.4.2.1. Creating a Single Intel Quartus Prime Project for a Standard Incremental Compilation Flow

Use the *<top-level project>***.tcl** file that contains the Synplify assignments for the top-level design. This method allows you to import all of the partitions into one Intel Quartus Prime project and optimize all modules within the project at once, taking advantage of the performance preservation and compilation time reduction offered by incremental compilation.

All of the constraints from the top-level project are passed to the Intel Quartus Prime software in the top-level **.tcl** file, but constraints made in the lower-level projects within the Synplify software are not forward-annotated. Enter these constraints manually in your Intel Quartus Prime project.

Figure 7. Design Flow Using Multiple .vqm Files with One Intel Quartus Prime Project

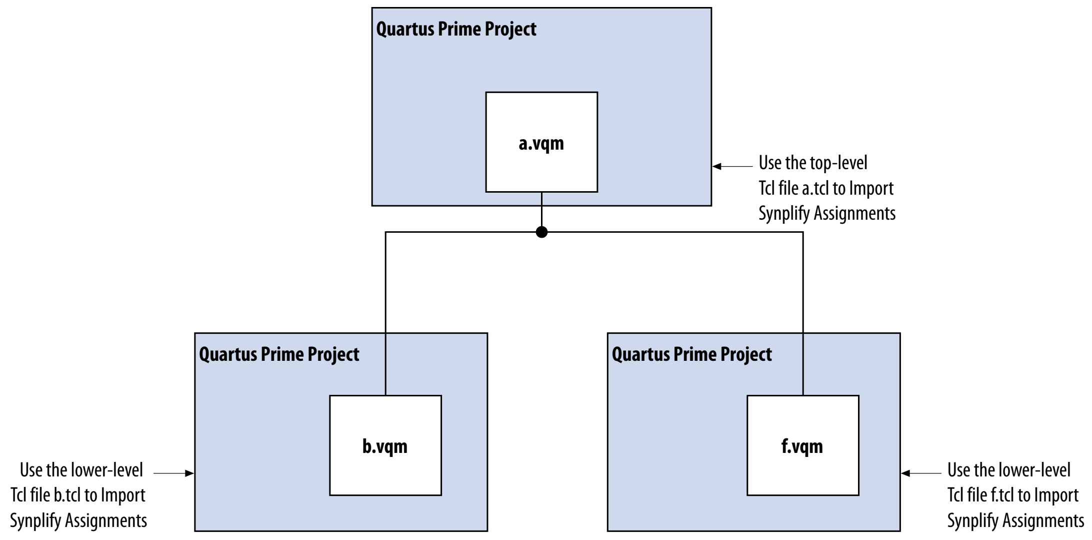

Quartus Prime Project
a.vqm
Use a.tcl to import top-level Synplify Pro assignments. Enter any lower-level assignments manually.
b.vqm
f.vqm

1.11.4.2.2. Creating Multiple Intel Quartus Prime Projects for a Bottom-Up Incremental Compilation Flow

Use the **.tcl** file that is created for each **.vqm** file by the Synplify software for each Synplify project. This method generates multiple Intel Quartus Prime projects, one for each block in the design. The designers in the project can optimize their own blocks separately within the Intel Quartus Prime software and export the placement of their own blocks.

Designers should create a LogicLock region to create a design floorplan for each block to avoid conflicts between partitions. The top-level designer then imports all the blocks and assignments into the top-level project. This method allows each block in the design to be optimized separately and then imported into one top-level project.

Figure 8. Design Flow Using Multiple Synplify Projects and Multiple Intel Quartus Prime Projects


Quartus Prime Project
a.vqm
Use the top-level Tcl file a.tcl to Import Synplify Assignments
Quartus Prime Project
b.vqm
Use the lower-level Tcl file b.tcl to Import Synplify Assignments
Quartus Prime Project
f.vqm
Use the lower-level Tcl file f.tcl to Import Synplify Assignments

1.11.5. Performing Incremental Compilation in the Intel Quartus Prime Software

In a standard design flow using Multipoint Synthesis, the Synplify software uses the Intel Quartus Prime top-level **.tcl** file to ensure that the two tools databases stay synchronized. The Tcl file creates, changes, or deletes partition assignments in the Intel Quartus Prime software for Compile Points that you create, change, or delete in the Synplify software. However, if you create, change, or delete a partition in the Intel Quartus Prime software, the Synplify software does not change your Compile Point settings. Make any corresponding change in your Synplify project to ensure that you create the correct **.vqm** files.

Note: If you use the NativeLink integration feature, the Synplify software does not use any information about design partition assignments that you have set in the Intel Quartus Prime software.

If you create netlist files with multiple Synplify projects, or if you do not use the Synplify Pro or Premier-generated **.tcl** files to update constraints in your Intel Quartus Prime project, you must ensure that your Synplify **.vqm** netlists align with your Intel Quartus Prime partition settings.

After you have set up your Intel Quartus Prime project with **.vqm** netlist files as separate design partitions, set the appropriate Intel Quartus Prime options to preserve your compilation results. On the Assignments menu, click **Design Partitions Window**. Change the **Netlist Type** to **Post-Fit** to preserve the previous compilation's post-fit placement results. If you do not make these settings, the Intel Quartus Prime software does not reuse the placement or routing results from the previous compilation.

You can take advantage of incremental compilation with your Synplify design to reduce compilation time in the Intel Quartus Prime software and preserve the results for unchanged design blocks.

Related Information

Using the Intel Quartus Prime Software to Run the Synplify Software on page 8

1.12. Synopsys Synplify* Support Revision History

Date	Version	Changes
2016.05.03	16.0.0	• Noted limitations of NativeLink synthesis.
2015.11.02	15.1.0	• Changed instances of *Quartus II* to *Intel Quartus Prime*.
November 2013	13.1.0	Dita conversion. Restructured content.
June 2012	12.0.0	Removed survey link.
November 2011	10.1.1	Template update.
December 2010	10.1.0	• Changed to new document template. • Removed Classic Timing Analyzer support. • Removed the "altera_implement_in_esb or altera_implement_in_eab" section. • Edited the "Creating a Intel Quartus Prime Project for Compile Points and Multiple .vqm Files" on page 14–33 section for changes with the incremental compilation flow. • Edited the "Creating a Intel Quartus Prime Project for Multiple .vqm Files" on page 14–39 section for changes with the incremental compilation flow. • Editorial changes.
July 2010	10.0.0	• Minor updates for the Intel Quartus Prime software version 10.0 release.
November 2009	9.1.0	• Minor updates for the Intel Quartus Prime software version 9.1 release.
March 2009	9.0.0	• Added new section "Exporting Designs to the Intel Quartus Prime Software Using NativeLink Integration" on page 14–14. • Minor updates for the Intel Quartus Prime software version 9.0 release. • Chapter 10 was previously Chapter 9 in software version 8.1.
November 2008	8.1.0	• Changed to 8-1/2 x 11 page size • Changed the chapter title from "Synplicity Synplify & Synplify Pro Support" to "Synopsys Synplify Support" • Replaced references to Synplicity with references to Synopsys • Added information about Synplify Premier • Updated supported device list • Added SystemVerilog information to Figure 14–1
May 2008	8.0.0	• Updated supported device list • Updated constraint annotation information for the Timing Analyzer • Updated RAM and MAC constraint limitations • Revised Table 9–1 • Added new section "Changing Synplify's Default Behavior for Instantiated Altera Megafunctions" • Added new section "Instantiating Intellectual Property Using the MegaWizard Plug-In Manager and IP Toolbench" • Added new section "Including Files for Intel Quartus Prime Placement and Routing Only" • Added new section "Additional Considerations for Compile Points" • Removed section "Apply the LogicLock Attributes" • Modified Figure 9–4, 9–43, 9–47. and 9–48 • Added new section "Performing Incremental Compilation in the Intel Quartus Prime Software" • Numerous text changes and additions throughout the chapter • Renamed several sections • Updated "Referenced Documents" section

Related Information

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

2. Mentor Graphics Precision* Synthesis Support

2.1. About Precision RTL Synthesis Support

This manual delineates the support for the Mentor Graphics® Precision RTL Synthesis and Precision RTL Plus Synthesis software in the Intel Quartus Prime software, as well as key design flows, methodologies and techniques for improving your results for Intel devices. This manual assumes that you have set up, licensed, and installed the Precision Synthesis software and the Intel Quartus Prime software.

Note: You must set up, license, and install the Precision RTL Plus Synthesis software if you want to use the incremental synthesis feature for incremental compilation and block-based design.

To obtain and license the Precision Synthesis software, refer to the Mentor Graphics website. To install and run the Precision Synthesis software and to set up your work environment, refer to the *Precision Synthesis Installation Guide* in the Precision Manuals Bookcase. To access the Manuals Bookcase in the Precision Synthesis software, click **Help** and select **Open Manuals Bookcase**.

Related Information

Mentor Graphics website

2.2. Design Flow

The following steps describe a basic Intel Quartus Prime design flow using the Precision Synthesis software:

1. Create Verilog HDL or VHDL design files.
2. Create a project in the Precision Synthesis software that contains the HDL files for your design, select your target device, and set global constraints.
3. Compile the project in the Precision Synthesis software.
4. Add specific timing constraints, optimization attributes, and compiler directives to optimize the design during synthesis. With the design analysis and cross-probing capabilities of the Precision Synthesis software, you can identify and improve circuit area and performance issues using prelayout timing estimates.

   Note: For best results, Mentor Graphics recommends specifying constraints that are as close as possible to actual operating requirements. Properly setting clock and I/O constraints, assigning clock domains, and indicating false and multicycle paths guide the synthesis algorithms more accurately toward a suitable solution in the shortest synthesis time.
5. Synthesize the project in the Precision Synthesis software.
6. Create an Intel Quartus Prime project and import the following files generated by the Precision Synthesis software into the Intel Quartus Prime project:

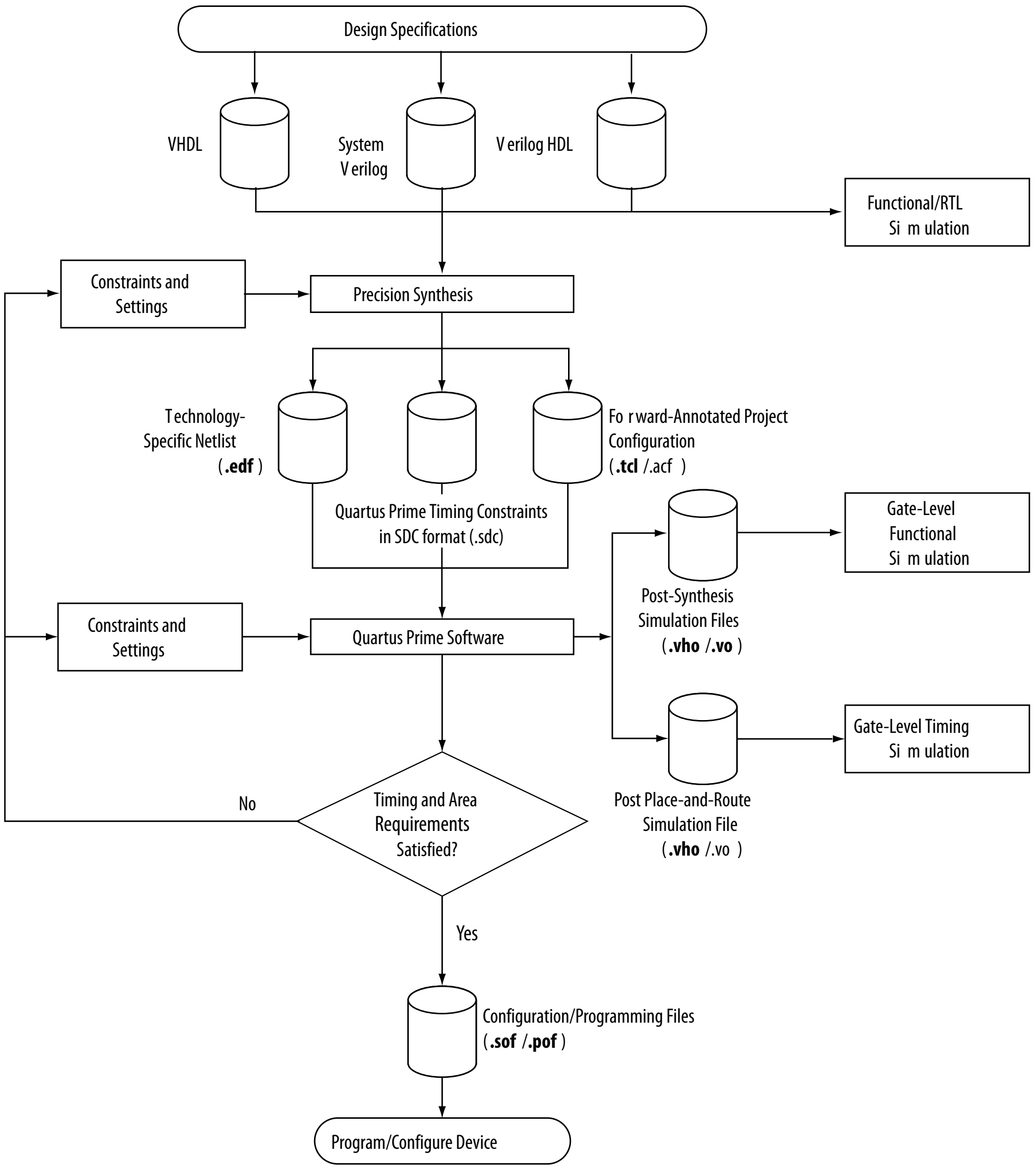
- The Verilog Quartus Mapping File (`.vqm`) netlist
- Synopsys Design Constraints File (`.sdc`) for Timing Analyzer constraints
- Tcl Script Files (`.tcl`) to set up your Intel Quartus Prime project and pass constraints

Note: If your design uses the Classic Timing Analyzer for timing analysis in the Intel Quartus Prime software versions 10.0 and earlier, the Precision Synthesis software generates timing constraints in the Tcl Constraints File (`.tcl`). If you are using the Intel Quartus Prime software versions 10.1 and later, you must use the Timing Analyzer for timing analysis.

7. After obtaining place-and-route results that meet your requirements, configure or program the Intel device.

You can run the Intel Quartus Prime software from within the Precision Synthesis software, or run the Precision Synthesis software using the Intel Quartus Prime software.

Figure 9. Design Flow Using the Precision Synthesis Software and Intel Quartus Prime Software


Design Specifications
VHDL
System Verilog
Verilog HDL
Functional/RTL Simulation
Constraints and Settings
Precision Synthesis
Technology-Specific Netlist (.edf)
Forward-Annotated Project Configuration (.tcl/.acf)
Quartus Prime Timing Constraints in SDC format (.sdc)
Gate-Level Functional Simulation
Post-Synthesis Simulation Files (.vho/.vo)
Constraints and Settings
Quartus Prime Software
Gate-Level Timing Simulation
No
Timing and Area Requirements Satisfied?
Post Place-and-Route Simulation File (.vho/.vo)
Yes
Configuration/Programming Files (.sof/.pof)
Program/Configure Device

Related Information

- Running the Intel Quartus Prime Software from within the Precision Synthesis Software on page 49
- Using the Intel Quartus Prime Software to Run the Precision Synthesis Software on page 50

2.2.1. Timing Optimization

If your area or timing requirements are not met, you can change the constraints and resynthesize the design in the Precision Synthesis software, or you can change the constraints to optimize the design during place-and-route in the Intel Quartus Prime software. Repeat the process until the area and timing requirements are met.

You can use other options and techniques in the Intel Quartus Prime software to meet area and timing requirements. For example, the **WYSIWYG Primitive Resynthesis** option can perform optimizations on your EDIF netlist in the Intel Quartus Prime software.

While simulation and analysis can be performed at various points in the design process, final timing analysis should be performed after placement and routing is complete.

2.3. Intel Device Family Support

The Precision Synthesis software supports active devices available in the current version of the Intel Quartus Prime software. Support for newly released device families may require an overlay. Contact Mentor Graphics for more information.

2.4. Precision Synthesis Generated Files

During synthesis, the Precision Synthesis software produces several intermediate and output files.

Table 4. Precision Synthesis Software Intermediate and Output Files

File Extension	File Description
`.psp`	Precision Synthesis Project File.
`.xdb`	Mentor Graphics Design Database File.
`.rep`(3)	Synthesis Area and Timing Report File.
`.vqm`(4)	Technology-specific netlist in `.vqm` file format. By default, the Precision Synthesis software creates `.vqm` files for Arria series, Cyclone series, and Stratix series devices. The Precision Synthesis software defaults to creating `.vqm` files when the device is supported.

continued...

(3) The timing report file includes performance estimates that are based on pre-place-and-route information. Use the f_{MAX} reported by the Intel Quartus Prime software after place-and-route for accurate post-place-and-route timing information. The area report file includes post-synthesis device resource utilization statistics that can differ from the resource usage after place-and-route due to black boxes or further optimizations performed during placement and routing. Use the device utilization reported by the Intel Quartus Prime software after place-and-route for final resource utilization results.

(4) The Precision Synthesis software-generated VQM file is supported by the Intel Quartus Prime software version 10.1 and later.

File Extension	File Description
`.tcl`	Forward-annotated Tcl assignments and constraints file. The *<project name>*`.tcl` file is generated for all devices. The `.tcl` file acts as the Intel Quartus Prime Project Configuration file and is used to make basic project and placement assignments, and to create and compile a Intel Quartus Prime project.
`.acf`	Assignment and Configurations file for backward compatibility with the MAX+PLUS II software. For devices supported by the MAX+PLUS II software, the MAX+PLUS II assignments are imported from the MAX+PLUS II `.acf` file.
`.sdc`	Intel Quartus Prime timing constraints file in Synopsys Design Constraints format. This file is generated automatically if the device uses the Timing Analyzer by default in the Intel Quartus Prime software, and has the naming convention *<project name>*`_pnr_constraints .sdc`.

Related Information

- Exporting Designs to the Intel Quartus Prime Software Using NativeLink Integration on page 49
- Synthesizing the Design and Evaluating the Results on page 48

2.5. Creating and Compiling a Project in the Precision Synthesis Software

After creating your design files, create a project in the Precision Synthesis software that contains the basic settings for compiling the design.

2.6. Mapping the Precision Synthesis Design

In the next steps, you set constraints and map the design to technology-specific cells. The Precision Synthesis software maps the design by default to the fastest possible implementation that meets your timing constraints. To accomplish this, you must specify timing requirements for the automatically determined clock sources. With this information, the Precision Synthesis software performs static timing analysis to determine the location of the critical timing paths. The Precision Synthesis software achieves the best results for your design when you set as many realistic constraints as possible. Be sure to set constraints for timing, mapping, false paths, multicycle paths, and other factors that control the structure of the implemented design.

Mentor Graphics recommends creating an `.sdc` file and adding this file to the **Constraint Files** section of the **Project Files** list. You can create this file with a text editor, by issuing command-line constraint parameters, or by directing the Precision Synthesis software to generate the file automatically the first time you synthesize your design. By default, the Precision Synthesis software saves all timing constraints and attributes in two files: `precision_rtl.sdc` and `precision_tech.sdc`. The `precision_rtl.sdc` file contains constraints set on the RTL-level database (post-compilation) and the `precision_tech.sdc` file contains constraints set on the gate-level database (post- synthesis) located in the current implementation directory.

You can also enter constraints at the command line. After adding constraints at the command line, update the `.sdc` file with the `update constraint file` command. You can add constraints that change infrequently directly to the HDL source files with HDL attributes or pragmas.

Note: The Precision `.sdc` file contains all the constraints for the Precision Synthesis project. For the Intel Quartus Prime software, placement constraints are written in a `.tcl` file and timing constraints for the Timing Analyzer are written in the Intel Quartus Prime`.sdc` file.

2.6.1. Setting Timing Constraints

The Precision Synthesis software uses timing constraints, based on the industry-standard `.sdc` file format, to deliver optimal results. Missing timing constraints can result in incomplete timing analysis and might prevent timing errors from being detected. The Precision Synthesis software provides constraint analysis prior to synthesis to ensure that designs are fully and accurately constrained. The *<project name>*`_pnr_constraints.sdc` file, which contains timing constraints in `.sdc` format, is generated in the Intel Quartus Prime software.

Note: Because the `.sdc` file format requires that timing constraints be set relative to defined clocks, you must specify your clock constraints before applying any other timing constraints.

You also can use multicycle path and false path assignments to relax requirements or exclude nodes from timing requirements, which can improve area utilization and allow the software optimizations to focus on the most critical parts of the design.

For details about the syntax of Synopsys Design Constraint commands, refer to the *Precision RTL Synthesis User's Manual* and the *Precision Synthesis Reference Manual*.

2.6.2. Setting Mapping Constraints

Mapping constraints affect how your design is mapped into the target Intel device. You can set mapping constraints in the user interface, in HDL code, or with the `set_attribute` command in the constraint file.

2.6.3. Assigning Pin Numbers and I/O Settings

The Precision Synthesis software supports assigning device pin numbers, I/O standards, drive strengths, and slew rate settings to top-level ports of the design. You can set these timing constraints with the `set_attribute` command, the GUI, or by specifying synthesis attributes in your HDL code. These constraints are forward-annotated in the *<project name>*`.tcl` file that is read by the Intel Quartus Prime software during place-and-route and do not affect synthesis.

You can use the `set_attribute` command in the Precision Synthesis software `.sdc` file to specify pin number constraints, I/O standards, drive strengths, and slow slew-rate settings. The table below describes the format to use for entries in the Precision Synthesis software constraint file.

Table 5. Constraint File Settings

Constraint	Entry Format for Precision Constraint File
Pin number	`set_attribute -name PIN_NUMBER -value "<pin number>" -port <port name>`
I/O standard	`set_attribute -name IOSTANDARD -value "<I/O Standard>" -port <port name>`

continued...

Constraint	Entry Format for Precision Constraint File
Drive strength	`set_attribute -name DRIVE -value "<drive strength in mA>" -port <port name>`
Slew rate	`set_attribute -name SLEW -value "TRUE | FALSE" -port <port name>`

You also can use synthesis attributes or pragmas in your HDL code to make these assignments.

Example 19. Verilog HDL Pin Assignment

```
//pragma attribute clk pin_number P10;
```

Example 20. VHDL Pin Assignment

```
attribute pin_number : string
attribute pin_number of clk : signal is "P10";
```

You can use the same syntax to assign the I/O standard using the `IOSTANDARD` attribute, drive strength using the attribute `DRIVE`, and slew rate using the `SLEW` attribute.

For more details about attributes and how to set these attributes in your HDL code, refer to the *Precision Synthesis Reference Manual.*

2.6.4. Assigning I/O Registers

The Precision Synthesis software performs timing-driven I/O register mapping by default. You can force a register to the device IO element (IOE) using the Complex I/O constraint. This option does not apply if you turn off **I/O pad insertion.**

Note: You also can make the assignment by right-clicking on the pin in the Schematic Viewer.

For the Stratix series, Cyclone series, and the MAX II device families, the Precision Synthesis software can move an internal register to an I/O register without any restrictions on design hierarchy.

For more mature devices, the Precision Synthesis software can move an internal register to an I/O register only when the register exists in the top-level of the hierarchy. If the register is buried in the hierarchy, you must flatten the hierarchy so that the buried registers are moved to the top-level of the design.

2.6.5. Disabling I/O Pad Insertion

The Precision Synthesis software assigns I/O pad atoms (device primitives used to represent the I/O pins and I/O registers) to all ports in the top-level of a design by default. In certain situations, you might not want the software to add I/O pads to all I/O pins in the design. The Intel Quartus Prime software can compile a design without I/O pads; however, including I/O pads provides the Precision Synthesis software with more information about the top-level pins in the design.

2.6.5.1. Preventing the Precision Synthesis Software from Adding I/O Pads

If you are compiling a subdesign as a separate project, I/O pins cannot be primary inputs or outputs of the device; therefore, the I/O pins should not have an I/O pad associated with them.

To prevent the Precision Synthesis software from adding I/O pads:

- You can use the Precision Synthesis GUI or add the following command to the project file:

```
setup_design -addio=false
```

2.6.5.2. Preventing the Precision Synthesis Software from Adding an I/O Pad on an Individual Pin

To prevent I/O pad insertion on an individual pin when you are using a black box, such as DDR or a phase-locked loop (PLL), at the external ports of the design, perform the following steps:

1. Compile your design.
2. Use the Precision Synthesis GUI to select the individual pin and turn off I/O pad insertion.

Note: You also can make this assignment by attaching the `nopad` attribute to the port in the HDL source code.

2.6.6. Controlling Fan-Out on Data Nets

Fan-out is defined as the number of nodes driven by an instance or top-level port. High fan-out nets can cause significant delays that result in an unroutable net. On a critical path, high fan-out nets can cause longer delays in a single net segment that result in the timing constraints not being met. To prevent this behavior, each device family has a global fan-out value set in the Precision Synthesis software library. In addition, the Intel Quartus Prime software automatically routes high fan-out signals on global routing lines in the Intel device whenever possible.

To eliminate routability and timing issues associated with high fan-out nets, the Precision Synthesis software also allows you to override the library default value on a global or individual net basis. You can override the library value by setting a `max_fanout` attribute on the net.

2.7. Synthesizing the Design and Evaluating the Results

During synthesis, the Precision Synthesis software optimizes the compiled design, and then writes out netlists and reports to the implementation subdirectory of your working directory after the implementation is saved, using the following naming convention:

```
<project name>_impl_<number>
```

After synthesis is complete, you can evaluate the results for area and timing. The *Precision RTL Synthesis User's Manual* describes different results that can be evaluated in the software.

There are several schematic viewers available in the Precision Synthesis software: RTL schematic, Technology-mapped schematic, and Critical Path schematic. These analysis tools allow you to quickly and easily isolate the source of timing or area issues, and to make additional constraint or code changes to optimize the design.

2.7.1. Obtaining Accurate Logic Utilization and Timing Analysis Reports

Historically, designers have relied on post-synthesis logic utilization and timing reports to determine the amount of logic their design requires, the size of the device required, and how fast the design runs. However, today's FPGA devices provide a wide variety of advanced features in addition to basic registers and look-up tables (LUTs). The Intel Quartus Prime software has advanced algorithms to take advantage of these features, as well as optimization techniques to increase performance and reduce the amount of logic required for a given design. In addition, designs can contain black boxes and functions that take advantage of specific device features. Because of these advances, synthesis tool reports provide post-synthesis area and timing estimates, but you should use the place-and-route software to obtain final logic utilization and timing reports.

2.8. Exporting Designs to the Intel Quartus Prime Software Using NativeLink Integration

The NativeLink feature in the Intel Quartus Prime software facilitates the seamless transfer of information between the Intel Quartus Prime software and EDA tools, which allows you to run other EDA design entry/synthesis, simulation, and timing analysis tools automatically from within the Intel Quartus Prime software.

After a design is synthesized in the Precision Synthesis software, the technology-mapped design is written to the current implementation directory as an EDIF netlist file, along with a Intel Quartus Prime Project Configuration File and a place-and-route constraints file. You can use the Project Configuration script, <*project name*>.**tcl**, to create and compile a Intel Quartus Prime project for your EDIF or VQM netlist. This script makes basic project assignments, such as assigning the target device specified in the Precision Synthesis software. If you select a newer Intel device, the constraints are written in SDC format to the <*project name*>_**pnr_constraints.sdc** file by default, which is used by the Fitter and the Timing Analyzer in the Intel Quartus Prime software.

Use the following Precision Synthesis software command before compilation to generate the <*project name*>_**pnr_constraints.sdc**:

```
setup_design -timequest_sdc
```

With this command, the file is generated after synthesis.

2.8.1. Running the Intel Quartus Prime Software from within the Precision Synthesis Software

The Precision Synthesis software also has a built-in place-and-route environment that allows you to run the Intel Quartus Prime Fitter and view the results in the Precision Synthesis GUI. This feature is useful when performing an initial compilation of your design to view post-place-and-route timing and device utilization results. Not all the advanced Intel Quartus Prime options that control the compilation process are available when you use this feature.

Two primary Precision Synthesis software commands control the place-and-route process. Use the `setup_place_and_route` command to set the place-and-route options. Start the process with the `place_and_route` command.

Precision Synthesis software uses individual Intel Quartus Prime executables, such as analysis and synthesis, Fitter, and the Timing Analyzer for improved runtime and memory utilization during place and route. This flow is referred to as the **Intel Quartus Prime Modular** flow option in the Precision Synthesis software. By default, the Precision Synthesis software generates a Intel Quartus Prime Project Configuration File (**.tcl** file) for current device families. Timing constraints that you set during synthesis are exported to the Intel Quartus Prime place-and-route constraints file *<project name>*_**pnr_constraints.sdc**.

After you compile the design in the Intel Quartus Prime software from within the Precision Synthesis software, you can invoke the Intel Quartus Prime GUI manually and then open the project using the generated Intel Quartus Prime project file. You can view reports, run analysis tools, specify options, and run the various processing flows available in the Intel Quartus Prime software.

For more information about running the Intel Quartus Prime software from within the Precision Synthesis software, refer to the *Intel Quartus Prime Integration* chapter in the *Precision Synthesis Reference Manual*.

2.8.2. Running the Intel Quartus Prime Software Manually Using the Precision Synthesis-Generated Tcl Script

You can run the Intel Quartus Prime software using a Tcl script generated by the Precision Synthesis software. To run the Tcl script generated by the Precision Synthesis software to set up your project and start a full compilation, perform the following steps:

1. Ensure the **.vqm** file, **.tcl** files, and **.sdc** file are located in the same directory. The files should be located in the implementation directory by default.
2. In the Intel Quartus Prime software, on the View menu, point to **Utility Windows** and click **Tcl Console**.
3. At the Tcl Console command prompt, type the command:

```
source <path>/<project name>.tcl
```

4. On the File menu, click **Open Project**. Browse to the project name and click **Open**.
5. Compile the project in the Intel Quartus Prime software.

2.8.3. Using the Intel Quartus Prime Software to Run the Precision Synthesis Software

With NativeLink integration, you can set up the Intel Quartus Prime software to run the Precision Synthesis software. This feature allows you to use the Precision Synthesis software to synthesize a design as part of a standard compilation. When you use this feature, the Precision Synthesis software does not use any timing constraints or assignments that you have set in the Intel Quartus Prime software.

2.8.4. Passing Constraints to the Intel Quartus Prime Software

The place-and-route constraints script forward-annotates timing constraints that you made in the Precision Synthesis software. This integration allows you to enter these constraints once in the Precision Synthesis software, and then pass them automatically to the Intel Quartus Prime software.

The following constraints are translated by the Precision Synthesis software and are applicable to the Timing Analyzer:

- `create_clock`
- `set_input_delay`
- `set_output_delay`
- `set_max_delay`
- `set_min_delay`
- `set_false_path`
- `set_multicycle_path`

2.8.4.1. create_clock

You can specify a clock in the Precision Synthesis software.

Example 21. Specifying a Clock Using create_clock

```
create_clock -name <clock_name> -period <period in ns> \
-waveform {<edge_list>} -domain <ClockDomain> <pin>
```

The period is specified in units of nanoseconds (ns). If no clock domain is specified, the clock belongs to a default clock domain `main`. All clocks in the same clock domain are treated as synchronous (related) clocks. If no *<clock_name>* is provided, the default name `virtual_default` is used. The *<edge_list>* sets the rise and fall edges of the clock signal over an entire clock period. The first value in the list is a rising transition, typically the first rising transition after time zero. The waveform can contain any even number of alternating edges, and the edges listed should alternate between rising and falling. The position of any edge can be equal to or greater than zero but must be equal to or less than the clock period.

If `-waveform` *`<edge_list>`* is not specified and `-period` *`<period in ns>`* is specified, the default waveform has a rising edge of 0.0 and a falling edge of *<period_value>*/2.

The Precision Synthesis software maps the clock constraint to the Timing Analyzer `create_clock` setting in the Intel Quartus Prime software.

The Intel Quartus Prime software supports only clock waveforms with two edges in a clock cycle. If the Precision Synthesis software finds a multi-edge clock, it issues an error message when you synthesize your design in the Precision Synthesis software.

2.8.4.2. set_input_delay

This port-specific input delay constraint is specified in the Precision Synthesis software.

Example 22. Specifying set_input_delay

```
set_input_delay {<delay_value> <port_pin_list>} \
-clock <clock_name> -rise -fall -add_delay
```

This constraint is mapped to the `set_input_delay` setting in the Intel Quartus Prime software.

When the reference clock *<clock_name>* is not specified, all clocks are assumed to be the reference clocks for this assignment. The input pin name for the assignment can be an input pin name of a time group. The software can use the `clock_fall` option to specify delay relative to the falling edge of the clock.

Note: Although the Precision Synthesis software allows you to set input delays on pins inside the design, these constraints are not sent to the Intel Quartus Prime software, and a message is displayed.

2.8.4.3. set_output_delay

This port-specific output delay constraint is specified in the Precision Synthesis software.

Example 23. Using the set_output_delay Constraint

```
set_output_delay {<delay_value> <port_pin_list>} \
-clock <clock_name> -rise -fall -add_delay
```

This constraint is mapped to the `set_output_delay` setting in the Intel Quartus Prime software.

When the reference clock *<clock_name>* is not specified, all clocks are assumed to be the reference clocks for this assignment. The output pin name for the assignment can be an output pin name of a time group.

Note: Although the Precision Synthesis software allows you to set output delays on pins inside the design, these constraints are not sent to the Intel Quartus Prime software.

2.8.4.4. set_max_delay and set_min_delay

The maximum delay and minimum delay for a point-to-point timing path constraint is specified in the Precision Synthesis software.

Example 24. Using the set_max_delay Constraint

```
set_max_delay -from {<from_node_list>} -to {<to_node_list>} <delay_value>
```

Example 25. Using the set_min_delay Constraint

```
set_min_delay -from {<from_node_list>} -to {<to_node_list>} <delay_value>
```

The `set_max_delay` and `set_min_delay` commands specify that the maximum and minimum respectively, required delay for any start point in *<from_node_list>* to any endpoint in *<to_node_list>* must be less than or greater than *<delay_value>*. Typically, you use these commands to override the default setup constraint for any path with a specific maximum or minimum time value for the path.

The node lists can contain a collection of clocks, registers, ports, pins, or cells. The `-from` and `-to` parameters specify the source (start point) and the destination (endpoint) of the timing path, respectively. The source list (*<from_node_list>*) cannot include output ports, and the destination list (*<to_node_list>*) cannot include input ports. If you include more than one node on a list, you must enclose the nodes in quotes or in braces ({ }).

If you specify a clock in the source list, you must specify a clock in the destination list. Applying `set_max_delay` or `set_min_delay` setting between clocks applies the exception from all registers or ports driven by the source clock to all registers or ports driven by the destination clock. Applying exceptions between clocks is more efficient than applying them for specific node-to-node, or node-to-clock paths. If you want to specify pin names in the list, the source must be a clock pin and the destination must be any non-clock input pin to a register. Assignments from clock pins, or to and from cells, apply to all registers in the cell or for those driven by the clock pin.

2.8.4.5. set_false_path

The false path constraint is specified in the Precision Synthesis software.

Example 26. Using the set_false_path Constraint

```
set_false_path -to <to_node_list> -from <from_node_list> -reset_path
```

The node lists can be a list of clocks, ports, instances, and pins. Multiple elements in the list can be represented using wildcards such as * and ?.

In a place-and-route Tcl constraints file, this false path setting in the Precision Synthesis software is mapped to a `set_false_path` setting. The Intel Quartus Prime software supports `setup`, `hold`, `rise`, or `fall` options for this assignment.

The node lists for this assignment represents top-level ports and/or nets connected to instances (end points of timing assignments).

Any false path setting in the Precision Synthesis software can be mapped to a setting in the Intel Quartus Prime software with a `through` path specification.

2.8.4.6. set_multicycle_path

The multicycle path constraint is specified in the Precision Synthesis software.

Example 27. Using the set_multicycle_path Constraint

```
set_multicycle_path <multiplier_value> [-start] [-end] \
-to <to_node_list> -from <from_node_list> -reset_path
```

The node list can contain clocks, ports, instances, and pins. Multiple elements in the list can be represented using wildcards such as * and ?. Paths without multicycle path definitions are identical to paths with multipliers of 1. To add one additional cycle to the datapath, use a multiplier value of 2. The option `start` indicates that source clock cycles should be considered for the multiplier. The option `end` indicates that destination clock cycles should be considered for the multiplier. The default is to reference the end clock.

In the place-and-route Tcl constraints file, the multicycle path setting in the Precision Synthesis software is mapped to a `set_multicycle_path` setting. The Intel Quartus Prime software supports the `rise` or `fall` options on this assignment.

The node lists represent top-level ports and/or nets connected to instances (end points of timing assignments). The node lists can contain wildcards (such as `*`); the Intel Quartus Prime software automatically expands all wildcards.

Any multicycle path setting in Precision Synthesis software can be mapped to a setting in the Intel Quartus Prime software with a `-through` specification.

2.9. Guidelines for Intel FPGA IP Cores and Architecture-Specific Features

Intel provides parameterizable IP cores, including the LPMs, and device-specific Intel FPGA IP, and IP available through third-party partners. You can use IP cores by instantiating them in your HDL code or by inferring certain functions from generic HDL code.

If you want to instantiate an IP core such as a PLL in your HDL code, you can instantiate and parameterize the function using the port and parameter definitions, or you can customize a function with the parameter editor. Intel recommends using the IP Catalog and parameter editor, which provides a graphical interface within the Intel Quartus Prime software for customizing and parameterizing any available IP core for the design.

The Precision Synthesis software automatically recognizes certain types of HDL code and infers the appropriate IP core.

Related Information

Inferring Intel FPGA IP Cores from HDL Code on page 57

2.9.1. Instantiating IP Cores With IP Catalog-Generated Verilog HDL Files

The IP Catalog generates a Verilog HDL instantiation template file *<output file>*`_inst.v` and a hollow-body black box module declaration *<output file>*`_bb.v` for use in your Precision Synthesis design. Incorporate the instantiation template file, *<output file>*`_inst.v`, into your top-level design to instantiate the IP core wrapper file, *<output file>*`.v`.

Include the hollow-body black box module declaration *<output file>*`_bb.v` in your Precision Synthesis project to describe the port connections of the black box. Adding the IP core wrapper file *<output file>*.v in your Precision Synthesis project is optional, but you must add it to your Intel Quartus Prime project along with the Precision Synthesis generated EDIF or VQM netlist.

Alternatively, you can include the IP core wrapper file *<output file>*.v in your Precision Synthesis project and turn on the **Exclude file from Compile Phase** option in the Precision Synthesis software to exclude the file from compilation and to copy the file to the appropriate directory for use by the Intel Quartus Prime software during place-and-route.

2.9.2. Instantiating IP Cores With IP Catalog-Generated VHDL Files

The IP Catalog generates a VHDL component declaration file <*output file*>`.cmp` and a VHDL instantiation template file <*output file*>`_inst.vhd` for use in your Precision Synthesis design. Incorporate the component declaration and instantiation template into your top-level design to instantiate the IP core wrapper file, <*output file*>`.vhd`.

Adding the IP core wrapper file <*output file*>`.vhd` in your Precision Synthesis project is optional, but you must add the file to your Intel Quartus Prime project along with the Precision Synthesis-generated EDIF or VQM netlist.

Alternatively, you can include the IP core wrapper file <*output file*>`.v` in your Precision Synthesis project and turn on the **Exclude file from Compile Phase** option in the Precision Synthesis software to exclude the file from compilation and to copy the file to the appropriate directory for use by the Intel Quartus Prime software during place-and-route.

2.9.3. Instantiating Intellectual Property With the IP Catalog and Parameter Editor

Many Intel FPGA IP functions include a resource and timing estimation netlist that the Precision Synthesis software can use to synthesize and optimize logic around the IP efficiently. As a result, the Precision Synthesis software provides better timing correlation, area estimates, and Quality of Results (QoR) than a black box approach.

To create this netlist file, perform the following steps:

1. Select the IP function in the IP Catalog.
2. Click **Next** to open the Parameter Editor.
3. Click **Set Up Simulation**, which sets up all the EDA options.
4. Turn on the **Generate netlist** option to generate a netlist for resource and timing estimation and click **OK**.
5. Click **Generate** to generate the netlist file.

The Intel Quartus Prime software generates a file <*output file*>`_syn.v`. This netlist contains the "gray box" information for resource and timing estimation, but does not contain the actual implementation. Include this netlist file into your Precision Synthesis project as an input file. Then include the IP core wrapper file <*output file*>`.v|vhd` in the Intel Quartus Prime project along with your EDIF or VQM output netlist.

The generated "gray box" netlist file, <*output file*>`_syn.v` , is always in Verilog HDL format, even if you select VHDL as the output file format.

Note: For information about creating a gray box netlist file from the command line, search Altera's Knowledge Database.

2.9.4. Instantiating Black Box IP Functions With Generated Verilog HDL Files

You can use the `syn_black_box` or `black_box` compiler directives to declare a module as a black box. The top-level design files must contain the IP port mapping and a hollow-body module declaration. You can apply the directive to the module declaration in the top-level file or a separate file included in the project so that the Precision Synthesis software recognizes the module is a black box.

Note: The `syn_black_box` and `black_box` directives are supported only on module or entity definitions.

The example below shows a sample top-level file that instantiates `my_verilogIP.v`, which is a simplified customized variation generated by the IP Catalog and Parameter Editor.

Example 28. Top-Level Verilog HDL Code with Black Box Instantiation of IP

```
module top (clk, count);
   input clk;
   output[7:0] count;

   my_verilogIP verilogIP_inst (.clock (clk), .q (count));
endmodule

// Module declaration
// The following attribute is added to create a
// black box for this module.
module my_verilogIP (clock, q) /* synthesis syn_black_box */;
   input clock;
   output[7:0] q;
endmodule
```

2.9.5. Instantiating Black Box IP Functions With Generated VHDL Files

You can use the `syn_black_box` or `black_box` compiler directives to declare a component as a black box. The top-level design files must contain the IP core variation component declaration and port mapping. Apply the directive to the component declaration in the top-level file.

Note: The `syn_black_box` and `black_box` directives are supported only on module or entity definitions.

The example below shows a sample top-level file that instantiates `my_vhdlIP.vhd`, which is a simplified customized variation generated by the IP Catalog and Parameter Editor.

Example 29. Top-Level VHDL Code with Black Box Instantiation of IP

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY top IS
  PORT (
    clk: IN STD_LOGIC ;
    count: OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
  );
END top;

ARCHITECTURE rtl OF top IS
  COMPONENT my_vhdlIP
  PORT (
    clock: IN STD_LOGIC ;
    q: OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
  );
  end COMPONENT;
  attribute syn_black_box : boolean;
  attribute syn_black_box of my_vhdlIP: component is true;
  BEGIN
    vhdlIP_inst : my_vhdlIP PORT MAP (
       clock => clk,
       q => count
    );
END rtl;
```

2.9.6. Inferring Intel FPGA IP Cores from HDL Code

The Precision Synthesis software automatically recognizes certain types of HDL code and maps arithmetical operators, relational operators, and memory (RAM and ROM), to technology-specific implementations. This functionality allows technology-specific resources to implement these structures by inferring the appropriate Intel function to provide optimal results. In some cases, the Precision Synthesis software has options that you can use to disable or control inference.

For coding style recommendations and examples for inferring technology-specific architecture in Intel devices, refer to the *Precision Synthesis Style Guide*.

2.9.6.1. Multipliers

The Precision Synthesis software detects multipliers in HDL code and maps them directly to device atoms to implement the multiplier in the appropriate type of logic. The Precision Synthesis software also allows you to control the device resources that are used to implement individual multipliers.

2.9.6.1.1. Controlling DSP Block Inference for Multipliers

By default, the Precision Synthesis software uses DSP blocks available in Stratix series devices to implement multipliers. The default setting is **AUTO**, which allows the Precision Synthesis software to map to logic look-up tables (LUTs) or DSP blocks, depending on the size of the multiplier. You can use the Precision Synthesis GUI or HDL attributes for direct mapping to only logic elements or to only DSP blocks.

Table 6. Options for dedicated_mult Parameter to Control Multiplier Implementation in Precision Synthesis

Value	Description
ON	Use only DSP blocks to implement multipliers, regardless of the size of the multiplier.
OFF	Use only logic (LUTs) to implement multipliers, regardless of the size of the multiplier.
AUTO	Use logic (LUTs) or DSP blocks to implement multipliers, depending on the size of the multipliers.

2.9.6.2. Setting the Use Dedicated Multiplier Option

To set the `Use Dedicated Multiplier` option in the Precision Synthesis GUI, compile the design, and then in the Design Hierarchy browser, right-click the operator for the desired multiplier and click **Use Dedicated Multiplier**.

2.9.6.3. Setting the dedicated_mult Attribute

To control the implementation of a multiplier in your HDL code, use the `dedicated_mult` attribute with the appropriate value as shown in the examples below.

Example 30. Setting the dedicated_mult Attribute in Verilog HDL

```
//synthesis attribute <signal name> dedicated_mult <value>
```

Example 31. Setting the dedicated_mult Attribute in VHDL

```
ATTRIBUTE dedicated_mult: STRING;
ATTRIBUTE dedicated_mult OF <signal name>: SIGNAL IS <value>;
```

The `dedicated_mult` attribute can be applied to signals and wires; it does not work when applied to a register. This attribute can be applied only to simple multiplier code, such as `a = b * c`.

Some signals for which the `dedicated_mult` attribute is set can be removed during synthesis by the Precision Synthesis software for design optimization. In such cases, if you want to force the implementation, you should preserve the signal by setting the `preserve_signal` attribute to TRUE.

Example 32. Setting the preserve_signal Attribute in Verilog HDL

```
//synthesis attribute <signal name> preserve_signal TRUE
```

Example 33. Setting the preserve_signal Attribute in VHDL

```
ATTRIBUTE preserve_signal: BOOLEAN;
ATTRIBUTE preserve_signal OF <signal name>: SIGNAL IS TRUE;
```

Example 34. Verilog HDL Multiplier Implemented in Logic

```
module unsigned_mult (result, a, b);
   output [15:0] result;
   input [7:0] a;
   input [7:0} b;
   assign result = a * b;
   //synthesis attribute result dedicated_mult OFF
endmodule
```

Example 35. VHDL Multiplier Implemented in Logic

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;
USE ieee.std_logic_unsigned.ALL;

ENTITY unsigned_mult IS
   PORT(
      a: IN std_logic_vector (7 DOWNTO 0);
      b: IN std_logic_vector (7 DOWNTO 0);
      result: OUT std_logic_vector (15 DOWNTO 0));
ATTRIBUTE dedicated_mult: STRING;
END unsigned_mult;

ARCHITECTURE rtl OF unsigned_mult IS
   SIGNAL a_int, b_int: UNSIGNED (7 downto 0);
   SIGNAL pdt_int: UNSIGNED (15 downto 0);
ATTRIBUTE dedicated_mult OF pdt_int: SIGNAL IS "OFF;
```

```
BEGIN
   a_int <= UNSIGNED (a);
   b_int <= UNSIGNED (b);
   pdt_int <= a_int * b_int;
   result <= std_logic_vector(pdt_int);
END rtl;
```

2.9.6.4. Multiplier-Accumulators and Multiplier-Adders

The Precision Synthesis software also allows you to control the device resources used to implement multiply-accumulators or multiply-adders in your project or in a particular module.

The Precision Synthesis software detects multiply-accumulators or multiply-adders in HDL code and infers an ALTMULT_ACCUM or ALTMULT_ADD IP cores so that the logic can be placed in DSP blocks, or the software maps these functions directly to device atoms to implement the multiplier in the appropriate type of logic.

Note: The Precision Synthesis software supports inference for these functions only if the target device family has dedicated DSP blocks.

For more information about DSP blocks in Intel devices, refer to the appropriate Intel device family handbook and device-specific documentation. For details about which functions a given DSP block can implement, refer to the DSP Solutions Center on the Altera website.

For more information about inferring multiply-accumulator and multiply-adder IP cores in HDL code, refer to the Intel *Recommended HDL Coding Styles* and the Mentor Graphics *Precision Synthesis Style Guide*.

Related Information

Altera DSP Solutions website

2.9.6.5. Controlling DSP Block Inference

By default, the Precision Synthesis software infers the ALTMULT_ADD or ALTMULT_ACCUM IP cores appropriately in your design. These IP cores allow the Intel Quartus Prime software to select either logic or DSP blocks, depending on the device utilization and the size of the function.

You can use the `extract_mac` attribute to prevent inference of an ALTMULT_ADD or ALTMULT_ACCUM IP cores in a certain module or entity.

Table 7. Options for extract_mac Attribute Controlling DSP Implementation

Value	Description
TRUE	The ALTMULT_ADD or ALTMULT_ACCUM IP core is inferred.
FALSE	The ALTMULT_ADD or ALTMULT_ACCUM IP core is not inferred.

To control inference, use the `extract_mac` attribute with the appropriate value from the examples below in your HDL code.

Example 36. Setting the extract_mac Attribute in Verilog HDL

```
//synthesis attribute <module name> extract_mac <value>
```

Example 37. Setting the extract_mac Attribute in VHDL

```
ATTRIBUTE extract_mac: BOOLEAN;
ATTRIBUTE extract_mac OF <entity name>: ENTITY IS <value>;
```

To control the implementation of the multiplier portion of a multiply-accumulator or multiply-adder, you must use the `dedicated_mult` attribute.

You can use the `extract_mac`, `dedicated_mult`, and `preserve_signal` attributes (in Verilog HDL and VHDL) to implement the given DSP function in logic in the Intel Quartus Prime software.

Example 38. Using extract_mac, dedicated_mult, and preserve_signal in Verilog HDL

```
module unsig_altmult_accum1 (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_accum1 extract_mac FALSE
endmodule
```

Example 39. Using extract_mac, dedicated_mult, and preserve_signal in VHDL

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_signed.all;
ENTITY signedmult_add IS
   PORT(
      a, b, c, d: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
      result: OUT STD_LOGIC_VECTOR (15 DOWNTO 0));
   ATTRIBUTE preserve_signal: BOOLEANS;
   ATTRIBUTE dedicated_mult: STRING;
   ATTRIBUTE extract_mac: BOOLEAN;
   ATTRIBUTE extract_mac OF signedmult_add: ENTITY IS FALSE;
END signedmult_add;
ARCHITECTURE rtl OF signedmult_add IS
   SIGNAL a_int, b_int, c_int, d_int : signed (7 DOWNTO 0);
   SIGNAL pdt_int, pdt2_int : signed (15 DOWNTO 0);
   SIGNAL result_int: signed (15 DOWNTO 0);
   ATTRIBUTE preserve_signal OF pdt_int: SIGNAL IS TRUE;
   ATTRIBUTE dedicated_mult OF pdt_int: SIGNAL IS "OFF";
   ATTRIBUTE preserve_signal OF pdt2_int: SIGNAL IS TRUE;
   ATTRIBUTE dedicated_mult OF pdt2_int: SIGNAL IS "OFF";
BEGIN
   a_int <= signed (a);
   b_int <= signed (b);
```

```
    c_int <= signed (c);
    d_int <= signed (d);
    pdt_int <= a_int * b_int;
    pdt2_int <= c_int * d_int;
    result_int <= pdt_int + pdt2_int;
    result <= STD_LOGIC_VECTOR(result_int);
END rtl;
```

2.9.6.6. RAM and ROM

The Precision Synthesis software detects memory structures in HDL code and converts them to an operator that infers an ALTSYNCRAM or LPM_RAM_DP IP cores, depending on the device family. The software then places these functions in memory blocks.

The software supports inference for these functions only if the target device family has dedicated memory blocks.

For more information about inferring RAM and ROM IP cores in HDL code, refer to the *Precision Synthesis Style Guide*.

2.10. Incremental Compilation and Block-Based Design

As designs become more complex and designers work in teams, a block-based incremental design flow is often an effective design approach. In an incremental compilation flow, you can make changes to one part of the design while maintaining the placement and performance of unchanged parts of the design. Design iterations can be made dramatically faster by focusing new compilations on particular design partitions and merging results with the results of previous compilations of other partitions. You can perform optimization on individual blocks and then integrate them into a final design and optimize the design at the top-level.

The first step in an incremental design flow is to make sure that different parts of your design do not affect each other. You must ensure that you have separate netlists for each partition in your design. If the whole design is in one netlist file, changes in one partition affect other partitions because of possible node name changes when you resynthesize the design.

You can create different implementations for each partition in your Precision Synthesis project, which allows you to switch between partitions without leaving the current project file. You can also create a separate project for each partition if you require separate projects for a team-based design flow. Alternatively, you can use the incremental synthesis capability in the Precision RTL Plus software.

2.10.1. Creating a Design with Precision RTL Plus Incremental Synthesis

The Precision RTL Plus incremental synthesis flow for Intel Quartus Prime incremental compilation uses a partition-based approach to achieve faster design cycle time.

Using the incremental synthesis feature, you can create different netlist files for different partitions of a design hierarchy within one partition implementation, which makes each partition independent of the others in an incremental compilation flow. Only the portions of a design that have been updated must be recompiled during design iterations. You can make changes and resynthesize one partition in a design to create a new netlist without affecting the synthesis results or fitting of other partitions.

The following steps show a general flow for partition-based incremental synthesis with Intel Quartus Prime incremental compilation:

1. Create Verilog HDL or VHDL design files.
2. Determine which hierarchical blocks you want to treat as separate partitions in your design, and designate the partitions with the `incr_partition` attribute.
3. Create a project in the Precision RTL Plus Synthesis software and add the HDL design files to the project.
4. Enable incremental synthesis in the Precision RTL Plus Synthesis software using one of these methods:
   - Use the Precision RTL Plus Synthesis GUI to turn on **Enable Incremental Synthesis**.
   - Run the following command in the Transcript Window:

     ```
     setup_design -enable_incr_synth
     ```
5. Run the basic Precision Synthesis flow of compilation, synthesis, and place-and-route on your design. In subsequent runs, the Precision RTL Plus Synthesis software processes only the parts of the design that have changed, resulting in a shorter iteration than the initial run. The performance of the unchanged partitions is preserved.

   The Precision RTL Plus Synthesis software sets the netlist types of the unchanged partitions to **Post Fit** and the changed partitions to **Post Synthesis**. You can change the netlist type during timing closure in the Intel Quartus Prime software to obtain the best QoR.
6. Import the EDIF or VQM netlist for each partition and the top-level **.tcl** file into the Intel Quartus Prime software, and set up the Intel Quartus Prime project to use incremental compilation.
7. Compile your Intel Quartus Prime project.
8. If you want, you can change the Intel Quartus Prime incremental compilation netlist type for a partition with the **Design Partitions Window**. You can change the **Netlist Type** to one of the following options:
   - To preserve the previous post-fit placement results, change the **Netlist Type** of the partition to **Post-Fit**.
   - To preserve the previous routing results, set the **Fitter Preservation Level** of the partition to **Placement and Routing**.

2.10.1.1. Creating Partitions with the incr_partition Attribute

Partitions are set using the HDL `incr_partition` attribute. The Precision Synthesis software creates or deletes partitions by reading this attribute during compilation iterations. The attribute can be attached to either the design unit definition or an instance.

To delete partitions, you can remove the attribute or set the attribute value to false.

Note: The Precision Synthesis software ignores partitions set in a black box.

Example 40. Using incr_partition Attribute to Create a Partition in Verilog HDL

```
Design unit partition:

module my_block(
   input clk;
   output reg [31:0] data_out) /* synthesis incr_partition */ ;

Instance partition:

my_block my_block_inst(.clk(clk), .data_out(data_out));
// synthesis attribute my_block_inst incr_partition true
```

Example 41. Using incr_partition Attribute to a Create Partition in VHDL

```
Design unit partition:

entity my_block is
   port(
      clk : in std_logic;
      data_out : out std_logic_vector(31 downto 0)
   );
   attribute incr_partition : boolean;
   attribute incr_partition of my_block : entity is true;
end entity my_block;

Instance partition:

component my_block is
   port(
      clk : in std_logic;
      data_out : out std_logic_vector(31 downto 0)
   );
end component;

attribute incr_partition : boolean;
attribute incr_partition of my_block_inst : label is true;

my_block_inst my_block
   port map(clk, data_out);
```

2.10.2. Creating Multiple Mapped Netlist Files With Separate Precision Projects or Implementations

You can manually generate multiple netlist files, which can be VQM or EDIF files, for incremental compilation using black boxes and separate Precision projects or implementations for each design partition. This manual flow is supported in versions of the Precision software that do not include the incremental synthesis feature. You might also use this feature if you perform synthesis in a team-based environment without a top-level synthesis project that includes all of the lower-level design blocks.

In the Precision Synthesis software, create a separate implementation, or a separate project, for each lower-level module and for the top-level design that you want to maintain as a separate netlist file. Implement black box instantiations of lower-level modules in your top-level implementation or project.

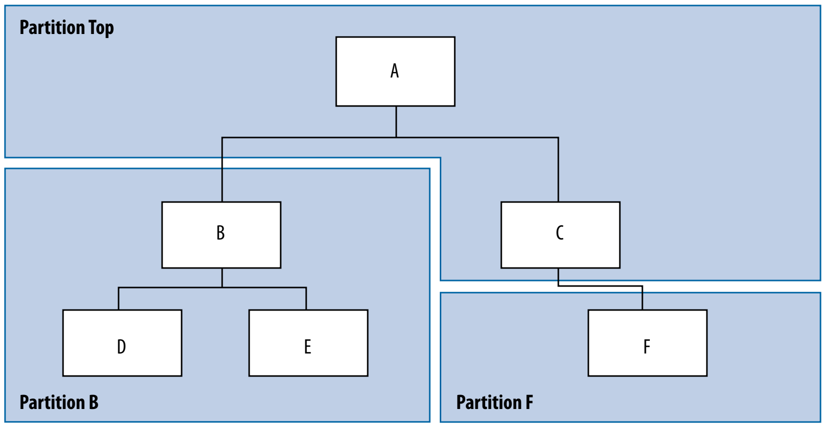
For more information about managing implementations and projects, refer to the *Precision RTL Synthesis User's Manual*.

Note: In a standard Intel Quartus Prime incremental compilation flow, Precision Synthesis software constraints made on lower-level modules are not passed to the Intel Quartus Prime software. Ensure that appropriate constraints are made in the top-level Precision Synthesis project, or in the Intel Quartus Prime project.

2.10.3. Creating Black Boxes to Create Netlists

In the figure below, the top-level partition contains the top-level block in the design (block A) and the logic that is not defined as part of another partition. In this example, the partition for top-level block A also includes the logic in the sub-block C. Because block F is contained in its own partition, it is not treated as part of the top-level partition A. Another separate partition, B, contains the logic in blocks B, D, and E. In a team-based design, different engineers may work on the logic in different partitions. One netlist is created for the top-level module A and its submodule C, another netlist is created for module B and its submodules D and E, while a third netlist is created for module F.

Figure 10. Partitions in a Hierarchical Design



To create multiple EDIF netlist files for this design, follow these steps:

1. Generate a netlist file for module B. Use **B.v**/.**vhd**, **D.v**/.**vhd**, and **E.v**/.**vhd** as the source files.
2. Generate a netlist file for module F. Use **F.v**/.**vhd** as the source file.
3. Generate a top-level netlist file for module A. Use **A.v**/.**vhd** and **C.v**/.**vhd** as the source files. Ensure that you create black boxes for modules B and F, which were optimized separately in the previous steps.

The goal is to individually synthesize and generate a netlist file for each lower-level module and then instantiate these modules as black boxes in the top-level file. You can then synthesize the top-level file to generate the netlist file for the top-level design. Finally, both the lower-level and top-level netlist files are provided to your Intel Quartus Prime project.

Note: When you make design or synthesis optimization changes to part of your design, resynthesize only the changed partition to generate the new netlist file. Do not resynthesize the implementations or projects for the unchanged partitions.

2.10.3.1. Creating Black Boxes in Verilog HDL

Any design block that is not defined in the project or included in the list of files to be read for a project is treated as a black box by the software. In Verilog HDL, you must provide an empty module declaration for any module that is treated as a black box.

A black box for the top-level file **A.v** is shown in the following example. Provide an empty module declaration for any lower-level files, which also contain a black box for any module beneath the current level of hierarchy.

Example 42. Verilog HDL Black Box for Top-Level File A.v

```
module A (data_in, clk, e, ld, data_out);
   input data_in, clk, e, ld;
   output [15:0] data_out;
   wire [15:0] cnt_out;
   B U1 (.data_in (data_in),.clk(clk), .ld (ld),.data_out(cnt_out));
   F U2 (.d(cnt_out), .clk(clk), .e(e), .q(data_out));
   // Any other code in A.v goes here.
endmodule
//Empty Module Declarations of Sub-Blocks B and F follow here.
// These module declarations (including ports) are required for black boxes.
module B (data_in, clk, ld, data_out);
   input data_in, clk, ld;
   output [15:0] data_out;
endmodule
module F (d, clk, e, q);
   input [15:0] d;
   input clk, e;
   output [15:0] q;
endmodule
```

2.10.3.2. Creating Black Boxes in VHDL

Any design block that is not defined in the project or included in the list of files to be read for a project is treated as a black box by the software. In VHDL, you must provide a component declaration for the black box.

A black box for the top-level file **A.vhd** is shown in the example below. Provide a component declaration for any lower-level files that also contain a black box or for any block beneath the current level of hierarchy.

Example 43. VHDL Black Box for Top-Level File A.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY A IS
   PORT ( data_in : IN INTEGER RANGE 0 TO 15;
      clk, e, ld : IN STD_LOGIC;
      data_out : OUT INTEGER RANGE 0 TO 15);
END A;
ARCHITECTURE a_arch OF A IS
COMPONENT B PORT(
   data_in : IN INTEGER RANGE 0 TO 15;
   clk, ld : IN STD_LOGIC;
   d_out : OUT INTEGER RANGE 0 TO 15);
END COMPONENT;
COMPONENT F PORT(
   d : IN INTEGER RANGE 0 TO 15;
   clk, e: IN STD_LOGIC;
   q : OUT INTEGER RANGE 0 TO 15);
END COMPONENT;
-- Other component declarations in A.vhd go here
signal cnt_out : INTEGER RANGE 0 TO 15;
BEGIN
   U1 : B
   PORT MAP (
      data_in => data_in,
      clk => clk,
      ld => ld,
      d_out => cnt_out);
   U2 : F
```

```
    PORT MAP (
       d => cnt_out,
       clk => clk,
       e => e,
       q => data_out);
    -- Any other code in A.vhd goes here
END a_arch;
```

After you complete the steps outlined above, you have different netlist files for each partition of the design. These files are ready for use with incremental compilation in the Intel Quartus Prime software.

2.10.4. Creating Intel Quartus Prime Projects for Multiple Netlist Files

The Precision Synthesis software creates a **.tcl** file for each implementation, and provides the Intel Quartus Prime software with the appropriate constraints and information to set up a project. When using incremental synthesis, the Precision RTL Plus Synthesis software creates only a single **.tcl** file, *<project name>***_incr_partitions.tcl**, to pass the partition information to the Intel Quartus Prime software.

Depending on your design methodology, you can create one Intel Quartus Prime project for all netlists, or a separate Intel Quartus Prime project for each netlist. In the standard incremental compilation design flow, you create design partition assignments for each partition in the design within a single Intel Quartus Prime project. This methodology provides the best QoR and performance preservation during incremental changes to your design. You might require a bottom-up design flow if each partition must be optimized separately, such as for third-party IP delivery.

To follow this design flow in the Intel Quartus Prime software, create separate Intel Quartus Prime projects and export each design partition and incorporate it into a top-level design using the incremental compilation features to maintain placement results.

Related Information

Running the Intel Quartus Prime Software Manually Using the Precision Synthesis-Generated Tcl Script on page 50

2.10.4.1. Creating a Single Intel Quartus Prime Project for a Standard Incremental Compilation Flow

Use the *<top-level project>***.tcl** file generated for the top-level partition to create your Intel Quartus Prime project and import all the netlists into this one Intel Quartus Prime project for an incremental compilation flow. You can optimize all partitions within the single Intel Quartus Prime project and take advantage of the performance preservation and compilation time reduction that incremental compilation provides.

All the constraints from the top-level implementation are passed to the Intel Quartus Prime software in the top-level **.tcl** file, but any constraints made only in the lower-level implementations within the Precision Synthesis software are not forward-annotated. Enter these constraints manually in your Intel Quartus Prime project.

2.10.4.2. Creating Multiple Intel Quartus Prime Projects for a Bottom-Up Flow

Use the **.tcl** files generated by the Precision Synthesis software for each Precision Synthesis software implementation or project to generate multiple Intel Quartus Prime projects, one for each partition in the design. Each designer in the project can optimize their block separately in the Intel Quartus Prime software and export the placement of their blocks using incremental compilation. Designers should create a LogicLock region to provide a floorplan location assignment for each block; the top-level designer should then import all the blocks and assignments into the top-level project.

2.10.5. Hierarchy and Design Considerations

To ensure the proper functioning of the synthesis flow, you can create separate partitions only for modules, entities, or existing netlist files. In addition, each module or entity must have its own design file. If two different modules are in the same design file, but are defined as being part of different partitions, incremental synthesis cannot be maintained because both regions must be recompiled when you change one of the modules.

Intel recommends that you register all inputs and outputs of each partition. This makes logic synchronous and avoids any delay penalty on signals that cross partition boundaries.

If you use boundary tri-states in a lower-level block, the Precision Synthesis software pushes the tri-states through the hierarchy to the top-level to make use of the tri-state drivers on output pins of Intel devices. Because pushing tri-states requires optimizing through hierarchies, lower-level tri-states are not supported with a block-based compilation methodology. You should use tri-state drivers only at the external output pins of the device and in the top-level block in the hierarchy.

2.11. Mentor Graphics Precision* Synthesis Support Revision History

Date	Version	Changes
2015.11.02	15.1.0	• Changed instances of *Quartus II* to *Intel Quartus Prime*.
June 2014	14.0.0	• Dita conversion. • Removed obsolete devices. • Replaced Intel FPGA IP, MegaWizard, and IP Toolbench content with IP Catalog and Parameter Editor content.
June 2012	12.0.0	• Removed survey link.
November 2011	10.1.1	• Template update. • Minor editorial changes.
December 2010	10.1.0	• Changed to new document template. • Removed Classic Timing Analyzer support. • Added support for **. vqm** netlist files. • Edited the "Creating Intel Quartus Prime Projects for Multiple EDIF Files" on page 15–30 section for changes with the incremental compilation flow. • Editorial changes.
July 2010	10.0.0	• Minor updates for the Intel Quartus Prime software version 10.0 release
November 2009	9.1.0	• Minor updates for the Intel Quartus Prime software version 9.1 release

continued...

Date	Version	Changes
March 2009	9.0.0	• Updated list of supported devices for the Intel Quartus Prime software version 9.0 release • Chapter 11 was previously Chapter 10 in software version 8.1
November 2008	8.1.0	• Changed to 8-1/2 x 11 page size • Title changed to *Mentor Graphics Precision Synthesis Support* • Updated list of supported devices • Added information about the Precision RTL Plus incremental synthesis flow • Updated Figure 10-1 to include SystemVerilog • Updated "Guidelines for Intel FPGA IP and Architecture-Specific Features" on page 10–19 • Updated "Incremental Compilation and Block-Based Design" on page 10–28 • Added section "Creating Partitions with the incr_partition Attribute" on page 10–29
May 2008	8.0.0	• Removed Mercury from the list of supported devices • Changed Precision version to 2007a update 3 • Added note for Stratix IV support • Renamed "Creating a Project and Compiling the Design" section to "Creating and Compiling a Project in the Precision RTL Synthesis Software" • Added information about constraints in the Tcl file • Updated document based on the Intel Quartus Prime software version 8.0

Related Information

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

A. Intel Quartus Prime Standard Edition User Guides

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

Related Information

- Intel Quartus Prime Standard Edition User Guide: Getting Started
  Introduces the basic features, files, and design flow of the Intel Quartus Prime Standard Edition software, including managing Intel Quartus Prime Standard Edition projects and IP, initial design planning considerations, and project migration from previous software versions.
- Intel Quartus Prime Standard Edition User Guide: Platform Designer
  Describes creating and optimizing systems using Platform Designer (Standard), a system integration tool that simplifies integrating customized IP cores in your project. Platform Designer (Standard) automatically generates interconnect logic to connect intellectual property (IP) functions and subsystems.
- Intel Quartus Prime Standard Edition User Guide: Design Recommendations
  Describes best design practices for designing FPGAs with the Intel Quartus Prime Standard Edition software. HDL coding styles and synchronous design practices can significantly impact design performance. Following recommended HDL coding styles ensures that Intel Quartus Prime Standard Edition synthesis optimally implements your design in hardware.
- Intel Quartus Prime Standard Edition User Guide: Design Compilation
  Describes set up, running, and optimization for all stages of the Intel Quartus Prime Standard Edition Compiler. The Compiler synthesizes, places, and routes your design before generating a device programming file.
- Intel Quartus Prime Standard Edition User Guide: Design Optimization
  Describes Intel Quartus Prime Standard Edition settings, tools, and techniques that you can use to achieve the highest design performance in Intel FPGAs. Techniques include optimizing the design netlist, addressing critical chains that limit retiming and timing closure, and optimization of device resource usage.
- Intel Quartus Prime Standard Edition User Guide: Programmer
  Describes operation of the Intel Quartus Prime Standard Edition Programmer, which allows you to configure Intel FPGA devices, and program CPLD and configuration devices, via connection with an Intel FPGA download cable.
- Intel Quartus Prime Standard Edition User Guide: Partial Reconfiguration
  Describes Partial Reconfiguration, an advanced design flow that allows you to reconfigure a portion of the FPGA dynamically, while the remaining FPGA design continues to function. Define multiple personas for a particular design region, without impacting operation in other areas.

- Intel Quartus Prime Standard Edition User Guide: Third-party Simulation
  Describes RTL- and gate-level design simulation support for third-party simulation tools by Aldec*, Cadence*, Mentor Graphics*, and Synopsys* that allow you to verify design behavior before device programming. Includes simulator support, simulation flows, and simulating Intel FPGA IP.
- Intel Quartus Prime Standard Edition User Guide: Third-party Synthesis
  Describes support for optional synthesis of your design in third-party synthesis tools by Mentor Graphics*, and Synopsys*. Includes design flow steps, generated file descriptions, and synthesis guidelines.
- Intel Quartus Prime Standard Edition User Guide: Debug Tools
  Describes a portfolio of Intel Quartus Prime Standard Edition in-system design debugging tools for real-time verification of your design. These tools provide visibility by routing (or "tapping") signals in your design to debugging logic. These tools include System Console, Signal Tap logic analyzer, Transceiver Toolkit, In-System Memory Content Editor, and In-System Sources and Probes Editor.
- Intel Quartus Prime Standard Edition User Guide: Timing Analyzer
  Explains basic static timing analysis principals and use of the Intel Quartus Prime Standard Edition Timing Analyzer, a powerful ASIC-style timing analysis tool that validates the timing performance of all logic in your design using an industry-standard constraint, analysis, and reporting methodology.
- Intel Quartus Prime Standard Edition User Guide: Power Analysis and Optimization
  Describes the Intel Quartus Prime Standard Edition Power Analysis tools that allow accurate estimation of device power consumption. Estimate the power consumption of a device to develop power budgets and design power supplies, voltage regulators, heat sink, and cooling systems.
- Intel Quartus Prime Standard Edition User Guide: Design Constraints
  Describes timing and logic constraints that influence how the Compiler implements your design, such as pin assignments, device options, logic options, and timing constraints. Use the Pin Planner to visualize, modify, and validate all I/O assignments in a graphical representation of the target device.
- Intel Quartus Prime Standard Edition User Guide: PCB Design Tools
  Describes support for optional third-party PCB design tools by Mentor Graphics* and Cadence*. Also includes information about signal integrity analysis and simulations with HSPICE and IBIS Models.
- Intel Quartus Prime Standard Edition User Guide: Scripting
  Describes use of Tcl and command line scripts to control the Intel Quartus Prime Standard Edition software and to perform a wide range of functions, such as managing projects, specifying constraints, running compilation or timing analysis, or generating reports.