


This chapter documents support for the Mentor Graphics® Precision RTL Synthesis and Precision RTL Plus Synthesis software in the Quartus® II software design flow, as well as key design methodologies and techniques for improving your results for Altera® devices.

The topics discussed in this chapter include:

- “Altera Device Family Support”
- “Design Flow” on page 16–2
- “Creating and Compiling a Project in the Precision Synthesis Software” on page 16–5
- “Mapping the Precision Synthesis Design” on page 16–5
- “Synthesizing the Design and Evaluating the Results” on page 16–9
- “Exporting Designs to the Quartus II Software Using NativeLink Integration” on page 16–10
- “Guidelines for Altera Megafunctions and Architecture-Specific Features” on page 16–15
- “Incremental Compilation and Block-Based Design” on page 16–24

This chapter assumes that you have set up, licensed, and installed the Precision Synthesis software and the Quartus II software. 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 at [www.mentor.com](http://www.mentor.com). 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**.

## Altera Device Family Support

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

The Precision Synthesis software also supports the FLEX 8000 and MAX 9000 legacy devices that are supported only in the Altera MAX+PLUS<sup>®</sup> II software, as well as ACEX<sup>®</sup> 1K, APEX<sup>™</sup> II, APEX 20K, APEX 20KC, APEX 20KE, FLEX<sup>®</sup> 10K, and FLEX 6000 legacy devices that are supported by the Quartus II software version 9.0 and earlier.

## Design Flow


The following steps describe a basic Quartus II 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. Refer to [“Creating and Compiling a Project in the Precision Synthesis Software” on page 16–5](#) for details.
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.



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. 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.
6. Create a Quartus II project and import the following files generated by the Precision Synthesis software into the Quartus II project:
  - The technology-specific EDIF (**.edf**) netlist or Verilog Quartus Mapping File (**.vqm**) netlist
  - Synopsys Design Constraints File (**.sdc**) for TimeQuest Timing Analyzer constraints

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

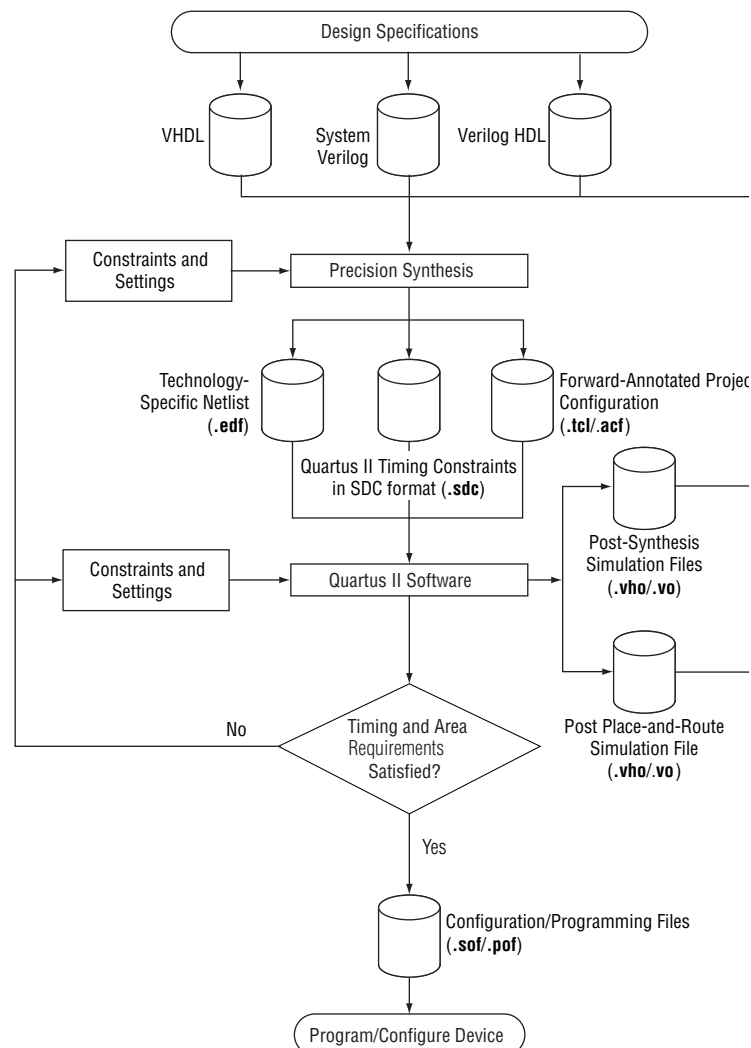
- Tcl Script Files (.tcl) to set up your Quartus II project and pass constraints

You can run the Quartus II software from within the Precision Synthesis software, or run the Precision Synthesis software using the Quartus II software. Refer to “Running the Quartus II Software from within the Precision Synthesis Software” on page 16–10 and “Using the Quartus II Software to Run the Precision Synthesis Software” on page 16–12 for more information.

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


Figure 16–1 shows the Quartus II design flow using the Precision Synthesis software as described in these steps, which are further described in detail in this chapter.

**Figure 16–1. Design Flow Using the Precision Synthesis Software and Quartus II Software**



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

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

 For more information about netlist optimizations, refer to the *Netlist Optimizations and Physical Synthesis* chapter in volume 2 of the *Quartus II Handbook*. For more recommendations about how to optimize your design, refer to the *Area and Timing Optimization* chapter in volume 2 of the *Quartus II Handbook*.

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.

During synthesis, the Precision Synthesis software produces several intermediate and output files, which are described in [Table 16-1](#).

**Table 16-1. Precision Synthesis Software Intermediate and Output Files**

File Extension	File Description
<b>.psp</b>	Precision Synthesis Project File.
<b>.xdb</b>	Mentor Graphics Design Database File.
<b>.rep</b> <sup>(1)</sup>	Synthesis Area and Timing Report File.
<b>.vqm/.edf</b> <sup>(2)</sup>	Technology-specific netlist in <b>.vqm</b> or <b>.edf</b> file format. By default, the Precision Synthesis software creates <b>.vqm</b> files for Arria series, Cyclone series, and Stratix series devices, and creates <b>.edf</b> files for ACEX, APEX, FLEX, and MAX series devices. The Precision Synthesis software can create <b>.edf</b> files for all Altera devices supported by the Quartus II software, but defaults to creating <b>.vqm</b> files when the device is supported.
<b>.tcl</b>	Forward-annotated Tcl assignments and constraints file. The <code>&lt;project name&gt;.tcl</code> file is generated for all devices. The <b>.tcl</b> file acts as the Quartus II Project Configuration file and is used to make basic project and placement assignments, and to create and compile a Quartus II project.
<b>.acf</b>	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 <b>.acf</b> file.
<b>.sdc</b>	Quartus II timing constraints file in Synopsys Design Constraints format. This file is generated automatically if the device uses the TimeQuest Timing Analyzer by default in the Quartus II software, and has the naming convention <code>&lt;project name&gt;_pnr_constraints.sdc</code> . For more information about generating a TimeQuest constraint file, refer to “ <a href="#">Exporting Designs to the Quartus II Software Using NativeLink Integration</a> ” on page 16-10.

**Notes to Table 16-1:**

- (1) The timing report file includes performance estimates that are based on pre-place-and-route information. Use the  $f_{MAX}$  reported by the Quartus II software after place-and-route for accurate post-place-and-route timing information. The area report file includes post-synthesis device resource utilization statistics that can differ from the resource usage after place-and-route due to black boxes or further optimizations performed during placement and routing. Use the device utilization reported by the Quartus II software after place-and-route for final resource utilization results. See “[Synthesizing the Design and Evaluating the Results](#)” on page 16-9 for details.
- (2) The Precision Synthesis software-generated VQM file is supported by the Quartus II software version 10.1 and later.

## 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.

To create a project, follow these steps:

1. In the Precision Synthesis software, click **New Project** in the **Design Bar** on the left side of the GUI.
2. Specify the **Project Name** and the **Project Folder**. The implementation name of the design corresponds to this project name.
3. Add input files to the project by clicking **Add Input Files** in the **Design Bar**. The Precision Synthesis software automatically detects the top-level module/entity of the design and uses it to name the current implementation directory, logs, reports, and netlist files.
4. In the **Design Bar**, click **Setup Design**.
5. To specify a target device family, expand **Altera** and select the target device and speed grade.
6. If you want, you can set a global design frequency and/or default input and output delays. This constrains all clock paths and I/O pins in your design. Modify the settings for individual paths or pins that do not require such a setting.
7. On the **Design Center** tab, right-click the **Output Files** folder and click **Output Options**.
8. To generate additional HDL netlists for post-synthesis simulation, select the desired output format. The Precision Synthesis software generates a separate file for each selected type of file: EDIF and Verilog HDL or VHDL.
9. To compile the design into a technology-independent implementation, in the **Design Bar**, click **Compile**.



## 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. To create a constraint file with the user interface, set constraints on design objects (such as clocks, design blocks, or pins) in the Design Hierarchy browser. 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.


-  The Precision **.sdc** file contains all the constraints for the Precision Synthesis project. For the Quartus II software, placement constraints are written in a **.tcl** file and timing constraints for the TimeQuest Timing Analyzer are written in the Quartus II **.sdc** file.
-  For details about the syntax of Synopsys Design Constraint commands, refer to the *Precision RTL Synthesis User's Manual* and the *Precision Synthesis Reference Manual*. For more details and examples of attributes, refer to the *Attributes* chapter in the *Precision Synthesis Reference Manual*. To access these manuals in the Precision Synthesis software, click **Help** and select **Open Manuals Bookcase**.

## Setting Timing Constraints

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

-  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*. To access these manuals in the Precision Synthesis software, click **Help** and select **Open Manuals Bookcase**.

## Setting Mapping Constraints

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

## Assigning Pin Numbers and I/O Settings

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

You can use the `set_attribute` command in the Precision Synthesis software `.sdc` file format to specify pin number constraints, I/O standards, drive strengths, and slow slew-rate settings. Table 16-2 describes the format to use for entries in the Precision Synthesis software constraint file.

**Table 16-2. Constraint File Settings**

Constraint	Entry Format for Precision Constraint File
Pin number	<code>set_attribute -name PIN_NUMBER -value "&lt;pin number&gt;" -port &lt;port name&gt;</code>
I/O standard	<code>set_attribute -name IOSTANDARD -value "&lt;I/O Standard&gt;" -port &lt;port name&gt;</code>
Drive strength	<code>set_attribute -name DRIVE -value "&lt;drive strength in mA&gt;" -port &lt;port name&gt;</code>
Slew rate	<code>set_attribute -name SLEW -value "TRUE   FALSE" -port &lt;port name&gt;</code>

You can also specify these options in the GUI. To specify a pin number or other I/O setting in the Precision Synthesis GUI, follow these steps:

1. After compiling the design, expand **Ports** in the Design Hierarchy Browser.
2. Under **Ports**, expand **Inputs** or **Outputs**.



You also can assign I/O settings by right-clicking the pin in the Schematic Viewer.

3. Right-click the desired pin name and select **Set Input Constraints** under **Inputs** or **Set Output Constraints** under **Outputs**.
4. Type the desired pin number on the Altera device in the **Pin Number** box in the **Port Constraints** dialog box.
5. Select the I/O standard from the **IO\_STANDARD** list.
6. For output pins, you can also select a drive strength setting and slew rate setting using the **DRIVE** and **SLOW SLEW** lists.

You also can use synthesis attributes or pragmas in your HDL code to make these assignments. Example 16-1 and Example 16-2 show code samples that make a pin assignment in your HDL code.


### Example 16-1. Verilog HDL Pin Assignment

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

### Example 16-2. VHDL Pin Assignment

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

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


 For more details about attributes and how to set these attributes in your HDL code, refer to the *Precision Synthesis Reference Manual*. To access this manual, in the Precision Synthesis software, click **Help** and select **Open Manuals Bookcase**.

## 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's IO element (IOE) using the Complex I/O constraint. This option does not apply if you turn off **I/O pad insertion**. Refer to [“Disabling I/O Pad Insertion” on page 16-8](#) for more information.

To force an I/O register into the device's IOE using the GUI, follow these steps:

1. After compiling the design, expand **Ports** in the Design Hierarchy browser.
2. Under **Ports**, expand **Inputs** or **Outputs**.
3. Under **Inputs** or **Outputs**, right-click the desired pin name, point to **Map Input Register to IO** or **Map Output Register to IO**, for input or output respectively, and click **True**.

 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.

## Disabling I/O Pad Insertion

The Precision Synthesis software assigns I/O pad atoms (device primitives used to represent the I/O pins and I/O registers) to all ports in the top-level of a design by default. In certain situations, you might not want the software to add I/O pads to all I/O pins in the design. The Quartus II 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.

### 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, perform the following steps:

1. On the Tools menu, click **Set Options**. The **Options** dialog box appears.

2. On the **Optimization** page, turn off **Add IO Pads**.
3. Click **Apply**.

These steps add the following command to the project file:

```
setup_design -addio=false
```

### 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. After compiling the design, in the Design Hierarchy browser, expand **Ports**.
2. Under **Ports**, expand **Inputs** or **Outputs**.
3. Under **Inputs** or **Outputs**, right-click the desired pin name and click **Set Input Constraints**.
4. In the **Port Constraints** dialog box for the selected pin name, turn off **Insert Pad**.



You also can make this assignment by right-clicking the pin in the Schematic Viewer or by attaching the `nopad` attribute to the port in the HDL source code.

## Controlling Fan-Out on Data Nets

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

## Synthesizing the Design and Evaluating the Results

To synthesize the design for the target device, click **Synthesize** in the **Precision Synthesis Design Bar**. 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* on the Mentor Graphics website 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.

## Obtaining Accurate Logic Utilization and Timing Analysis Reports

Historically, designers have relied on post-synthesis logic utilization and timing reports to determine the amount of logic their design requires, the size of the device required, and how fast the design runs. However, today's FPGA devices provide a wide variety of advanced features in addition to basic registers and look-up tables (LUTs). The Quartus II 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.

## Exporting Designs to the Quartus II Software Using NativeLink Integration

The NativeLink feature in the Quartus II software facilitates the seamless transfer of information between the Quartus II software and EDA tools, which allows you to run other EDA design entry/synthesis, simulation, and timing analysis tools automatically from within the Quartus II 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 Quartus II 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 Quartus II 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 an Arria GX, Stratix III, Cyclone III, or newer 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 TimeQuest Timing Analyzer in the Quartus II 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 the synthesis.

## Running the Quartus II 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 Quartus II 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, but not all the advanced Quartus II options that control the compilation process are available.

After you specify an Altera device as the target, set the options for the Quartus II software. On the Tools menu, click **Set Options**. On the **Integrated Place and Route** page, under **Quartus II Modular**, specify the path to the Quartus II executables in the **Path to Quartus II installation tree** box.

To automate the place-and-route process, click **Run Quartus II** in the **Quartus II Modular** window of the Precision Synthesis toolbar. The Quartus II software uses the current implementation directory as the Quartus II project directory and runs a full compilation in the background (that is, the user interface does not appear).

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

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



For more information about running the Quartus II software from within the Precision Synthesis software, refer to the *Altera Quartus II Integration* chapter in the *Precision Synthesis Reference Manual*. To access this manual in the Precision Synthesis software, click **Help** and select **Open Manuals Bookcase**.

## Running the Quartus II Software Manually Using the Precision Synthesis-Generated Tcl Script

You can run the Quartus II 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 `.edf` or `.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 Quartus II 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 Quartus II software.

## Using the Quartus II Software to Run the Precision Synthesis Software

With NativeLink integration, you can set up the Quartus II 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, such as incremental compilation partitions, that you have set in the Quartus II software.

- ❓ For detailed information about using NativeLink integration with the Precision Synthesis software, refer to *Using the NativeLink Feature with Other EDA Tools* in the Quartus II Help.

## Passing Constraints to the Quartus II 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 Quartus II software.

Refer to the introductory text in the section “[Exporting Designs to the Quartus II Software Using NativeLink Integration](#)” on page 16-10 for information on how to ensure the Precision Synthesis software targets the TimeQuest Timing Analyzer.

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

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

### **create\_clock**

You can specify a clock in the Precision Synthesis software, as shown in [Example 16-3](#).

#### **Example 16-3. 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 TimeQuest `create_clock` setting in the Quartus II software.

The Quartus II 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.

### set\_input\_delay

This port-specific input delay constraint is specified in the Precision Synthesis software, as shown in [Example 16-4](#).

#### Example 16-4. 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 Quartus II 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.



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

### set\_output\_delay

This port-specific output delay constraint is specified in the Precision Synthesis software, as shown in [Example 16-5](#).

#### Example 16-5. 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 Quartus II 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.



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

### set\_max\_delay and set\_min\_delay

The maximum delay for a point-to-point timing path constraint is specified in the Precision Synthesis software, as shown in [Example 16-6](#). The minimum delay for a point-to-point timing path constraint is shown in [Example 16-7](#).

#### Example 16-6. Using the set\_max\_delay Constraint

---

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

---

#### Example 16-7. 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.

### set\_false\_path

The false path constraint is specified in the Precision Synthesis software, as shown in [Example 16-8](#).

#### Example 16-8. 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 Quartus II 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 Quartus II software with a through path specification.

### **set\_multicycle\_path**

This multicycle path constraint is specified in the Precision Synthesis software, as shown in [Example 16-9](#).

#### **Example 16-9. 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 Quartus II 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 Quartus II software automatically expands all wildcards.

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

## **Guidelines for Altera Megafunctions and Architecture-Specific Features**

Altera provides parameterizable megafunctions, including the LPMs, device-specific Altera megafunctions, IP available as Altera MegaCore functions, and IP available through the Altera Megafunction Partners Program (AMPP<sup>SM</sup>). You can use megafunctions and IP functions by instantiating them in your HDL code or by inferring certain megafunctions from generic HDL code.

If you want to instantiate a megafunction 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 MegaWizard<sup>TM</sup> Plug-In Manager. Altera recommends using the MegaWizard Plug-In Manager, which provides a graphical interface within the Quartus II software for customizing and parameterizing any available megafunction for the design. [“Instantiating Altera Megafunctions Using the MegaWizard Plug-In Manager”](#) and [“Instantiating Intellectual Property With the MegaWizard Plug-In Manager and IP Toolbench”](#) on page 16-17 describe the MegaWizard Plug-In Manager flow with the Precision Synthesis software.

- For more information about specific Altera megafunctions and IP functions, refer to the [IP and Megafunctions](#) page of the Altera website.

The Precision Synthesis software automatically recognizes certain types of HDL code and infers the appropriate function. The Precision Synthesis software provides options to control inference of certain types of megafunctions, as described in [“Inferring Altera Megafunctions from HDL Code”](#) on page 16-19.

- For a detailed discussion about instantiating functions versus inferring functions to target Altera architecture-specific features, refer to the [Recommended HDL Coding Styles](#) chapter in volume 1 of the *Quartus II Handbook*. This chapter also provides details on using the MegaWizard Plug-In Manager in the Quartus II software and explains the files generated by the wizard, as well as coding style recommendations and HDL examples for inferring functions in Altera devices.

## Instantiating Altera Megafunctions Using the MegaWizard Plug-In Manager

This section describes how to instantiate Altera megafunctions with the MegaWizard Plug-In Manager, and how to generate the files that are included in the Precision Synthesis project for synthesis.

You can run the stand-alone version of the MegaWizard Plug-In Manager by typing the following command at a command prompt:

```
qmegawiz ←
```

### Instantiating Megafunctions With MegaWizard Plug-In Manager-Generated Verilog HDL Files

The MegaWizard Plug-In Manager 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 megafunction 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 megafunction wrapper file `<output file>.v` in your Precision Synthesis project is optional, but you must add it to your Quartus II project along with the Precision Synthesis-generated EDIF or VQM netlist.

Alternatively, you can include the megafunction wrapper file `<output file>.v` in your Precision Synthesis project and then right-click the file in the input file list, and select **Properties**. In the **Input file properties** dialog box, turn on **Exclude file from Compile Phase** and click **OK**. When this option is turned on, the Precision Synthesis software excludes the file from compilation and copies the file to the appropriate directory for use by the Quartus II software during place-and-route.

## Instantiating Megafunctions With MegaWizard Plug-In Manager-Generated VHDL Files

The MegaWizard Plug-In Manager 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 megafunction wrapper file, `<output file>.vhd`.

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

Alternatively, you can include the megafunction wrapper file `<output file>.vhd` in your Precision Synthesis project and then right-click the file in the input file list, and select **Properties**. In the **Input file properties** dialog box, turn on **Exclude file from Compile Phase** and click **OK**. When this option is turned on, the Precision Synthesis software excludes the file from compilation and copies the file to the appropriate directory for use by the Quartus II software during place-and-route.

## Instantiating Intellectual Property With the MegaWizard Plug-In Manager and IP Toolbench

Many Altera 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 MegaWizard Plug-In Manager.
2. Click **Next** to open the IP Toolbench.
3. Click **Set Up Simulation**, which sets up all the EDA options.
4. Turn on the **Generate netlist** option to generate a netlist for resource and timing estimation and click **OK**.
5. Click **Generate** to generate the netlist file.

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



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



There is currently no grey box support for SOPC Builder systems in the MegaWizard Plug-In Manager. For information about creating a grey box netlist file from the command line, search Altera's Knowledge Database. Alternatively, you can use a black box approach as described in [“Instantiating Black Box IP Functions With Generated Verilog HDL Files”](#).

## 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.



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

**Example 16-10** shows a sample top-level file that instantiates `my_verilogIP.v`, which is a simplified customized variation generated by the MegaWizard Plug-In Manager and IP Toolbench.

### Example 16-10. 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
```

## 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 megafunction variation component declaration and port mapping. Apply the directive to the component declaration in the top-level file.



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

**Example 16-11** shows a sample top-level file that instantiates `my_vhdlIP.vhd`, which is a simplified customized variation generated by the MegaWizard Plug-In Manager and IP Toolbench.

### Example 16-11. 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;
```

## Inferring Altera Megafunctions from HDL Code

The Precision Synthesis software automatically recognizes certain types of HDL code and maps arithmetical and relational operators, and memory (RAM and ROM), to efficient technology-specific implementations. This functionality allows technology-specific resources to implement these structures by inferring the appropriate Altera 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 Altera devices, refer to the *Recommended HDL Coding Styles* chapter in volume 1 of the *Quartus II Handbook*, and the *Precision Synthesis Style Guide* in the Precision Manuals Bookcase. To access these manuals, in the Precision Synthesis software, click **Help** and select **Open Manuals Bookcase**.

### 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, as described in the following section.

### 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 to direct mapping to only logic elements or to only DSP blocks.

The options for multiplier mapping in the Precision Synthesis software are described in [Table 16-3](#).

**Table 16-3. 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.

### 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**.

### 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 from [Table 16-3](#), as shown in [Example 16-12](#) and [Example 16-13](#).

#### Example 16-12. Setting the dedicated\_mult Attribute in Verilog HDL

---

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

---

#### Example 16-13. 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`, as shown in [Example 16-14](#) and [Example 16-15](#).

#### Example 16-14. Setting the preserve\_signal Attribute in Verilog HDL

---

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

---

---

### Example 16-15. Setting the `preserve_signal` Attribute in VHDL

---

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

---

Example 16-16 and Example 16-17 are examples, in Verilog HDL and VHDL, of using the dedicated `_mult` attribute to implement the given multiplier in regular logic in the Quartus II software.

---

### Example 16-16. 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 16-17. 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;
```

---

## 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` megafunction 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.

-  The Precision Synthesis software supports inference for these functions only if the target device family has dedicated DSP blocks. Refer to “[Controlling DSP Block Inference](#)” for more information.
-  For more information about DSP blocks in Altera devices, refer to the appropriate Altera 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 at [www.altera.com](http://www.altera.com).
-  For more information about inferring multiply-accumulator and multiply-adder megafunctions in HDL code, refer to the *Recommended HDL Coding Styles* chapter in volume 1 of the *Quartus II Handbook*, and the *Precision Synthesis Style Guide* in the Precision Synthesis Manuals Bookcase.

## Controlling DSP Block Inference

By default, the Precision Synthesis software infers the ALTMULT\_ADD or ALTMULT\_ACCUM megafunction appropriately in your design. These megafunctions allow the Quartus II 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 megafunction in a certain module or entity. The options for this attribute are described in [Table 16-4](#).

**Table 16-4. Options for `extract_mac` Attribute Controlling DSP Implementation**

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

To control inference, use the `extract_mac` attribute with the appropriate value from [Table 16-4](#) in your HDL code, as shown in [Example 16-18](#) and [Example 16-19](#).

### Example 16-18. Setting the `extract_mac` Attribute in Verilog HDL

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

### Example 16-19. 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.

[Example 16-20](#) and [Example 16-21](#) on [page 16-23](#) use the `extract_mac`, `dedicated_mult`, and `preserve_signal` attributes (in Verilog HDL and VHDL) to implement the given DSP function in logic in the Quartus II software.

**Example 16-20. 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 16-21. 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: BOOLEAN;
  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;

```

## 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 megafunction, 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 megafunctions in HDL code, refer to the *Recommended HDL Coding Styles* chapter in volume 1 of the *Quartus II Handbook*, and the *Precision Synthesis Style Guide* in the Precision Synthesis Manuals Bookcase. To access these manuals, in the Precision Synthesis software, click **Help** and select **Open Manuals Bookcase**.

## 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.

-  For more information about creating partitions and using incremental compilation in the Quartus II software, refer to the *Quartus II Incremental Compilation for Hierarchical and Team-Based Design* chapter in volume 1 of the *Quartus II Handbook*.

## Creating a Design with Precision RTL Plus Incremental Synthesis

The Precision RTL Plus incremental synthesis flow for Quartus II 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 Quartus II 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. For the syntax to create partitions, refer to “[Creating Partitions with the incr\\_partition Attribute](#)” on page 16–25.
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:
  - On the Tools menu, click **Set Options**. On the **Optimization** page, 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 Quartus II software to obtain the best QoR.

6. Import the EDIF or VQM netlist for each partition and the top-level `.tcl` file into the Quartus II software, and set up the Quartus II project to use incremental compilation.
7. Compile your Quartus II project.
8. If you want, you can change the Quartus II 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**.

### 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. [Example 16–22](#) and [Example 16–23](#) show how to use the attribute to create partitions.

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



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

---

#### Example 16-22. 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 16-23. 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);
```

---

## Creating Multiple Mapped Netlist Files With Separate Precision Projects or Implementations


This section describes how to 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*. To access this manual, in the Precision Synthesis software, click **Help** and select **Open Manuals Bookcase**.

When synthesizing the implementations for lower-level modules, perform these steps in the Precision Synthesis software:

1. On the Tools menu, turn off **Add IO Pads** on the **Optimization** page under **Set Options**.

 You must turn off the **Add IO Pads** option while synthesizing the lower-level modules individually. Enable the **Add IO Pads** option only while synthesizing the top-level module.


2. Read the HDL files for the modules.

 Modules can include black box instantiations of lower-level modules that are also maintained as separate netlist files.

3. Add constraints for all partitions in the design.

When synthesizing the top-level design implementation, perform these steps:

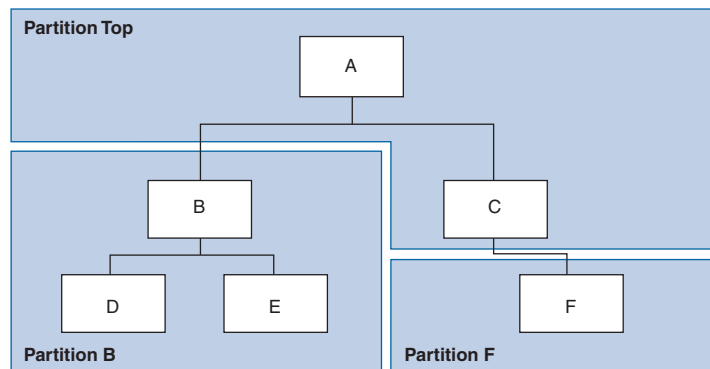
1. Read the HDL files for top-level designs.
2. On the Tools menu, click **Set Options**. On the **Optimization** page, turn on **Add IO Pads**.
3. Create black boxes for lower-level modules in the top-level design.
4. Add constraints.

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

## Creating Black Boxes to Create EDIF Netlists

This section describes how to create black boxes to create separate EDIF netlists. Figure 16-2 shows an example of a design hierarchy separated into various partitions.

**Figure 16-2. Partitions in a Hierarchical Design**



In Figure 16-2, 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. To create multiple EDIF netlist files for this design, follow these steps:

1. Generate an `.edf` file for module B. Use `B.v/.vhd`, `D.v/.vhd`, and `E.v/.vhd` as the source files.
2. Generate an `.edf` file for module F. Use `F.v/.vhd` as the source file.
3. Generate a top-level `.edf` 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 an `.edf` 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 `.edf` netlist file for the top-level design. Finally, both the lower-level and top-level `.edf` netlist files are provided to your Quartus II project.



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

### 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 16–24. 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
```

---

### 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 [Example 16–25](#). 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 16-25. 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 in this section, you have different EDIF netlist files for each partition of the design. These files are ready for use with incremental compilation in the Quartus II software.

## Creating Quartus II Projects for Multiple EDIF Files

The Precision Synthesis software creates a `.tcl` file for each implementation, and provides the Quartus II 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 Quartus II software. For details about using this Tcl script to set up your Quartus II project and to pass your top-level constraints, refer to [“Running the Quartus II Software Manually Using the Precision Synthesis-Generated Tcl Script”](#) on page 16-11.

Depending on your design methodology, you can create one Quartus II project for all EDIF netlists, or a separate Quartus II project for each EDIF netlist. In the standard incremental compilation design flow, you create design partition assignments for each partition in the design within a single Quartus II 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 Quartus II software, create separate Quartus II projects and export each design partition and incorporate it into a top-level design using the incremental compilation features to maintain placement results.

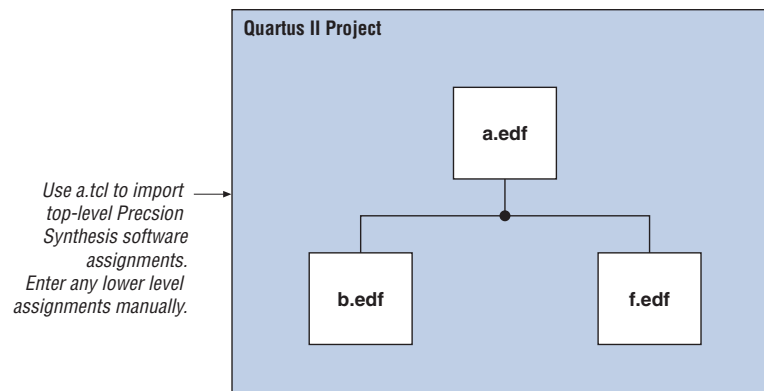
The following sections describe how to create the Quartus II projects for these two design flows.

### Creating a Single Quartus II Project for a Standard Incremental Compilation Flow

Use the `<top-level project>.tcl` file generated for the top-level partition to create your Quartus II project and import all the netlists into this one Quartus II project for an incremental compilation flow. You can optimize all partitions within the single Quartus II project and take advantage of the performance preservation and compilation time reduction that incremental compilation provides. Figure 16-3 shows the design flow for the example design in Figure 16-2 on page 16-28.

All the constraints from the top-level implementation are passed to the Quartus II 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 Quartus II project.

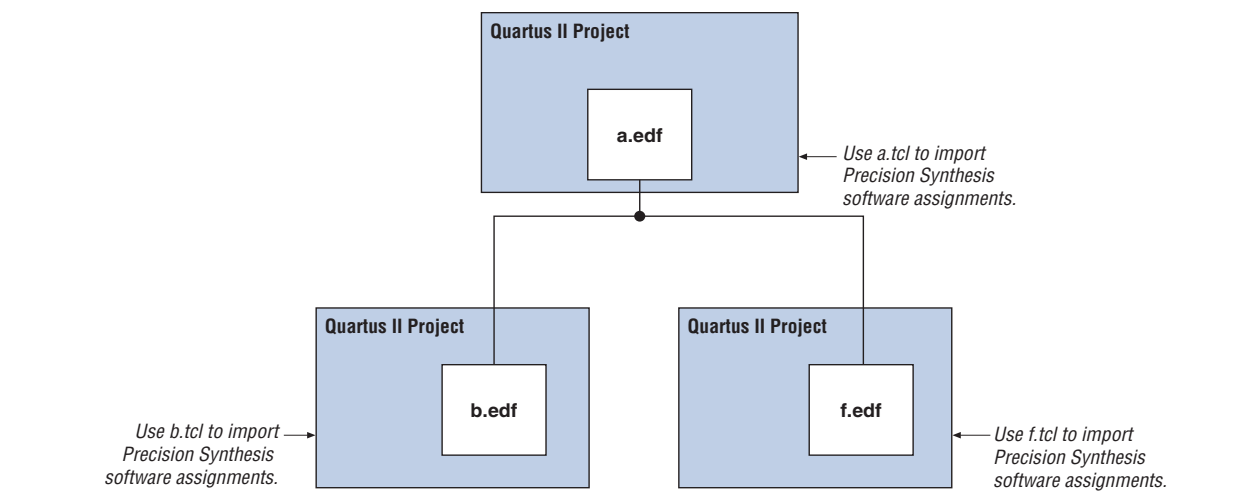
**Figure 16-3. Design Flow Using Multiple EDIF Files with One Quartus II Project**



## Creating Multiple Quartus II 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 Quartus II projects, one for each partition in the design. Each designer in the project can optimize their block separately in the Quartus II 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. Figure 16-4 shows the design flow for the example design in Figure 16-2 on page 16-28.

**Figure 16-4. Design Flow: Using Multiple EDIF Files with Multiple Quartus II Projects**




## 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.

Altera 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 Altera 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.

 For more tips on design partitioning, refer to the *Best Practices for Incremental Compilation Partitions and Floorplan Assignments* chapter in volume 1 of the *Quartus II Handbook*.

## Conclusion


The Mentor Graphics Precision Synthesis software and Quartus II design flow allow you to control how to prepare your design files for the Quartus II place-and-route process, which allows you to improve performance and optimizes your design for use with Altera devices. Several of the methodologies outlined in this chapter can help you optimize your design to achieve performance goals and decrease design time.


## Document Revision History

Table 16-5 shows the revision history for this document.

**Table 16-5. Document Revision History**

Date	Version	Changes
November 2011	10.1.1	<ul style="list-style-type: none"> <li>■ Template update.</li> <li>■ Minor editorial changes.</li> </ul>
December 2010	10.1.0	<ul style="list-style-type: none"> <li>■ Changed to new document template.</li> <li>■ Removed Classic Timing Analyzer support.</li> <li>■ Added support for <b>.vqm</b> netlist files.</li> <li>■ Edited the “Creating Quartus II Projects for Multiple EDIF Files” on page 15–30 section for changes with the incremental compilation flow.</li> <li>■ Editorial changes.</li> </ul>
July 2010	10.0.0	<ul style="list-style-type: none"> <li>■ Minor updates for the Quartus II software version 10.0 release</li> </ul>
November 2009	9.1.0	<ul style="list-style-type: none"> <li>■ Minor updates for the Quartus II software version 9.1 release</li> </ul>
March 2009	9.0.0	<ul style="list-style-type: none"> <li>■ Updated list of supported devices for the Quartus II software version 9.0 release</li> <li>■ Chapter 11 was previously Chapter 10 in software version 8.1</li> </ul>
November 2008	8.1.0	<ul style="list-style-type: none"> <li>■ Changed to 8-1/2 x 11 page size</li> <li>■ Title changed to <i>Mentor Graphics Precision Synthesis Support</i></li> <li>■ Updated list of supported devices</li> <li>■ Added information about the Precision RTL Plus incremental synthesis flow</li> <li>■ Updated Figure 10-1 to include SystemVerilog</li> <li>■ Updated “Guidelines for Altera Megafunctions and Architecture-Specific Features” on page 10–19</li> <li>■ Updated “Incremental Compilation and Block-Based Design” on page 10–28</li> <li>■ Added section “Creating Partitions with the <code>incr_partition</code> Attribute” on page 10–29</li> </ul>
May 2008	8.0.0	<ul style="list-style-type: none"> <li>■ Removed Mercury from the list of supported devices</li> <li>■ Changed Precision version to 2007a update 3</li> <li>■ Added note for Stratix IV support</li> <li>■ Renamed “Creating a Project and Compiling the Design” section to “Creating and Compiling a Project in the Precision RTL Synthesis Software”</li> <li>■ Added information about constraints in the Tcl file</li> <li>■ Updated document based on the Quartus II software version 8.0</li> </ul>

 For previous versions of the *Quartus II Handbook*, refer to the [Quartus II Handbook Archive](#).

 Take an [online survey](#) to provide feedback about this chapter.