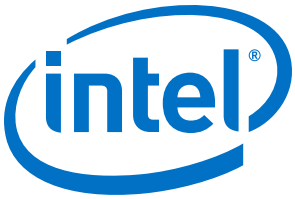


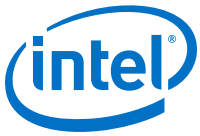
Intel® Quartus® Prime Pro Edition User Guide

Third-party Synthesis

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

UG-20138 | 2018.09.24
Latest document on the web: PDF | HTML

Contents



1. Mentor Graphics Precision* Synthesis Support

1.1. 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.

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

1.2. 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:

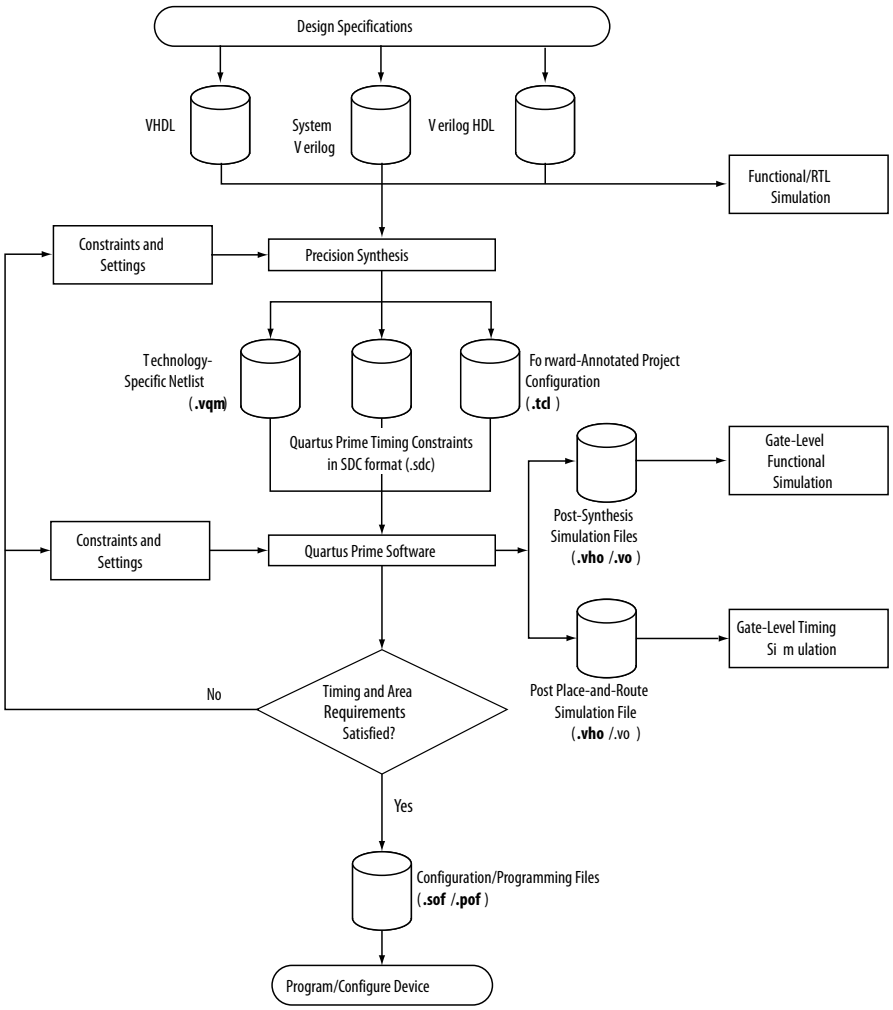
- The Verilog Quartus Mapping File (`.vqm`) netlist
- Synopsys Design Constraints File (`.sdc`) for Timing Analyzer constraints
- Tcl Script Files (`.tcl`) to set up your Intel Quartus Prime project and pass constraints

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



1.2.1. 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.

Related Information

- Netlist Optimizations and Physical Synthesis
- Timing Closure and Optimization

1.3. 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.

1.4. Precision Synthesis Generated Files

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

Table 1. Precision Synthesis Software Intermediate and Output Files

File Extension	File Description
`.psp`	Precision Synthesis Project File.
`.xdb`	Mentor Graphics Design Database File.
`.rep`(1)	Synthesis Area and Timing Report File.
`.vqm`(2)	Technology-specific netlist in `.vqm` file format.

continued...

(1) 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.

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

File Extension	File Description
	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.
`.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

Synthesizing the Design and Evaluating the Results on page 10

1.5. 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.

1.6. 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.

1.6.1. 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*.

1.6.2. 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.

1.6.3. 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 2. 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 1. Verilog HDL Pin Assignment

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

Example 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.*

1.6.4. 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.

1.6.5. 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.

1.6.5.1. 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
```

1.6.5.2. 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.

1.6.6. 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.

1.7. 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.

1.7.1. 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.

1.8. 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 14
- Recommended HDL Coding Styles, Design Recommendations User Guide

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

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

Include the hollow-body black box module declaration <*output file*>_bb.v in your Precision Synthesis project to describe the port connections of the black box. Adding the IP core wrapper file <*output file*>.v in your Precision Synthesis project is optional, but you must add it to your 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.

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

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

Adding the IP core wrapper file <*output file*>.vhd in your Precision Synthesis project is optional, but you must add the file to your 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.

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

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

To create this netlist file, perform the following steps:

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

The 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.

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

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

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

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

Example 3. 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
```

1.8.5. Instantiating Black Box IP Functions With Generated VHDL Files

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

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

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

Example 4. 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;
```

1.8.6. Inferring Intel FPGA IP Cores from HDL Code

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

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

Related Information

Recommended HDL Coding Styles, Design Recommendations User Guide

1.8.6.1. Multipliers

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

1.8.6.1.1. Controlling DSP Block Inference for Multipliers

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

Table 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.

1.8.6.2. Setting the Use Dedicated Multiplier Option

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

1.8.6.3. Setting the dedicated_mult Attribute

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

Example 5. Setting the dedicated_mult Attribute in Verilog HDL

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

Example 6. 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 7. Setting the preserve_signal Attribute in Verilog HDL

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

Example 8. Setting the preserve_signal Attribute in VHDL

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

Example 9. 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 10. 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;
```

1.8.6.4. Multiplier-Accumulators and Multiplier-Adders

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

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

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

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

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

Related Information

- Altera DSP Solutions website
- Intel Quartus Prime Pro Edition User Guide: Design Recommendations

1.8.6.5. Controlling DSP Block Inference

By default, the Precision Synthesis software infers the ALTMULT_ADD or ALTMULT_ACCUM IP cores appropriately in your design. These IP cores allow the 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 4. 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 11. Setting the extract_mac Attribute in Verilog HDL

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

Example 12. 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 13. Using extract_mac, dedicated_mult, and preserve_signal in Verilog HDL

```
module unsig_altmult_accuml (dataout, dataa, datab, clk, aclr, clken);
   input [7:0} dataa, datab;
   input clk, aclr, clken;
   output [31:0] dataout;

   reg    [31:0] dataout;
   wire   [15:0] multa;
   wire   [31:0] adder_out;

   assign multa = dataa * datab;

   //synthesis attribute multa preserve_signal TRUE
   //synthesis attribute multa dedicated_mult OFF
   assign adder_out = multa + dataout;

   always @ (posedge clk or posedge aclr)
   begin
      if (aclr)
      dataout <= 0;
      else if (clken)
      dataout <= adder_out;
   end

   //synthesis attribute unsig_altmult_accuml extract_mac FALSE
endmodule
```

Example 14. 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;
```

1.8.6.6. RAM and ROM

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

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

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

Related Information

Intel Quartus Prime Pro Edition User Guide: Design Recommendations

1.9. Mentor Graphics Precision* Synthesis Support Revision History

Document Version	Intel Quartus Prime Version	Changes
2018.09.24	18.1.0	Removed reference to obsolete .edf file from "Design Flow" diagram.
2018.05.07	18.0.0	Corrected trademark symbols on tool names.
2016.10.31	16.1.0	• Implemented Intel rebranding.

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.

continued...

Date	Version	Changes
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
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.

2. Synopsys Synplify* Support

2.1. 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.
- 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

2.2. Design Flow

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

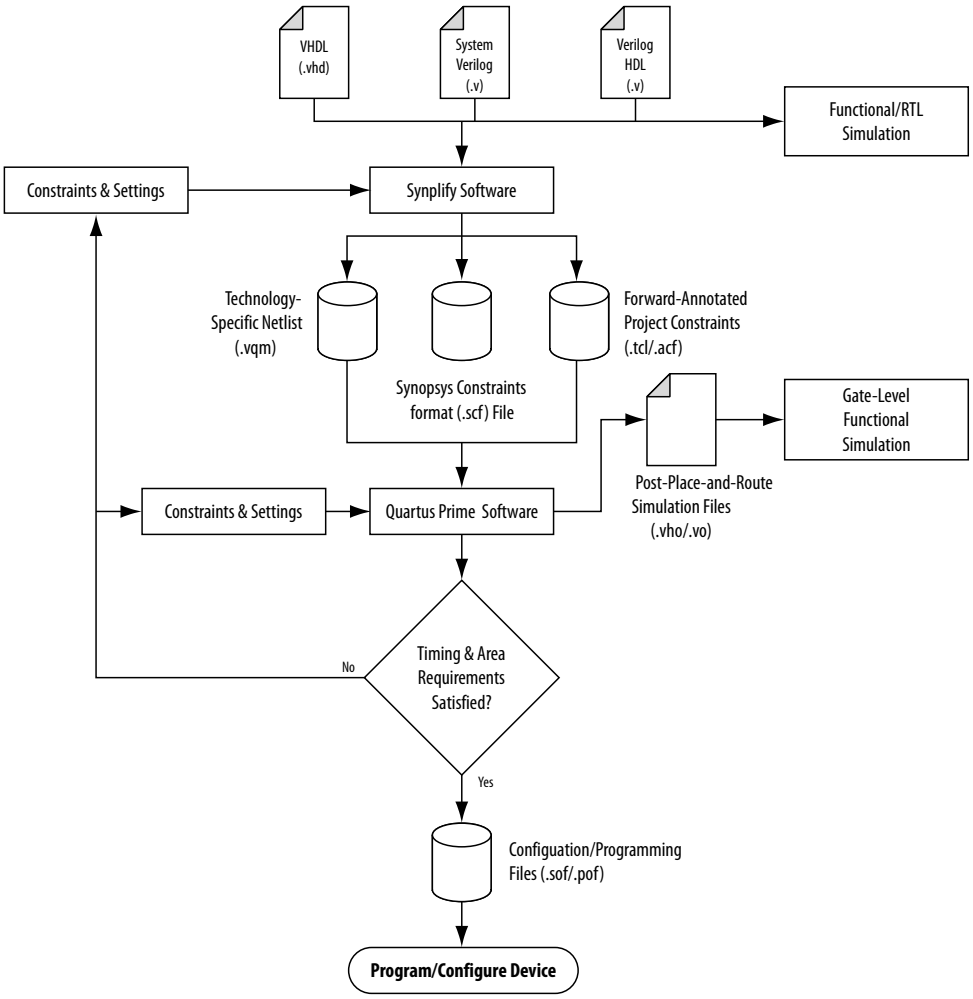
1. Create Verilog HDL (`.v`) or VHDL (`.vhd`) design files.
2. Set up a project in the Synplify software and add the HDL design files for synthesis.
3. Select a target device and add timing constraints and compiler directives in the Synplify software to help optimize the design during synthesis.
4. Synthesize 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 2. Recommended Design Flow



Related Information

- Synplify Software Generated Files on page 22
- Design Constraints Support on page 23

2.3. 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 32

2.4. Intel Device Family Support

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

Related Information

Synopsys Website

2.5. Tool Setup

2.5.1. 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 15. Specifying Intel Quartus Prime Software Version at the Command Line

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

2.6. Synplify Software Generated Files

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

Table 5. 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	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 .tclfile 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(3)	Synthesis Report file.

Related Information

Design Flow on page 20

2.7. 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.

Related Information

- Design Flow on page 20
- Synplify Optimization Strategies on page 25

(3) 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.

2.7.1. 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 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
```

2.7.2. 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 26

2.7.2.1. 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.

2.7.2.2. 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.

2.7.2.3. 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.

2.7.2.4. 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.

2.8. 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.

2.9. 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 23

2.9.1. 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.

2.9.2. 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.

2.9.3. 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 24

2.9.3.1. 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.

2.9.3.2. 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.

2.9.3.3. 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.

2.9.3.4. 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.

2.9.3.5. 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.

2.9.4. 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 6. `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 16. 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.

2.9.4.1. 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.

2.9.5. Optimization Attributes and Options

2.9.5.1. 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.

2.9.5.2. 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.

2.9.5.3. 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.

2.9.5.4. 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.

2.9.5.5. 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**.

2.9.5.6. 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**.

2.9.5.7. 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 17. Input and Output Register Delay

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

2.9.5.8. 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 */ ;
```

2.9.5.9. 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 18. 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.*

2.9.6. 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.

2.9.6.1. 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 19. 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";
```

2.9.6.2. 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**.

2.9.6.3. 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.

2.10. 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 22
- Recommended HDL Coding Styles, Design Recommendations User Guide
- About the IP Catalog Online Help

2.10.1. 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 22

2.10.1.1. 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.

2.10.1.2. 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.

2.10.1.3. 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 37
- 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.
- Generating a Netlist for 3rd Party Synthesis Tools online help

2.10.1.4. 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 37

2.10.1.5. 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 20. 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
```

2.10.1.6. 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 21. 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;
```

2.10.1.7. 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 22. 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*.

2.10.2. 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 23. 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"
```

2.10.3. Inferring Intel FPGA IP Cores from HDL Code

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

Related Information

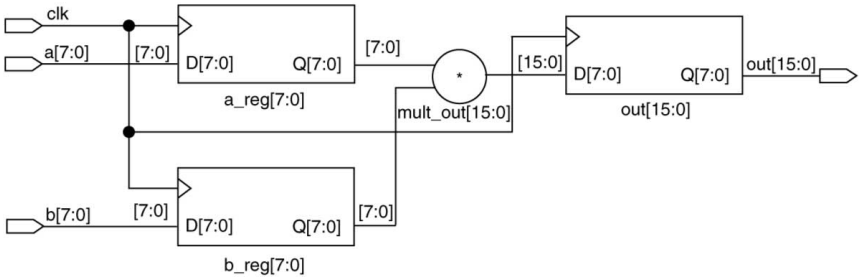
Recommended HDL Coding Styles, Design Recommendations User Guide

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



2.10.3.1.1. 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.

2.10.3.1.2. 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.

2.10.3.1.3. 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 7. 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 24. 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 25. 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;
```

2.10.3.2. 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 26. 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 27. 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;
```

2.10.3.3. 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 28. 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 29. 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";
```

2.10.3.4. 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.

2.10.3.5. 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.

2.11. Synopsys Synplify* Support Revision History

Document Version	Intel Quartus Prime Version	Changes
2018.09.24	18.1.0	Removed reference to obsolete .edf file from "Design Flow" diagram.
2018.05.07	18.0.0	Corrected trademark symbols on tool names.
2016.10.31	16.1.0	• Implemented Intel rebranding.

Date	Version	Changes
2016.05.03	16.0.0	• Removed support for NativeLink synthesis in Pro Edition
2015.11.02	15.1.0	• Changed instances of *Quartus II* to *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.

continued...

Date	Version	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.

A. Intel Quartus Prime Pro Edition User Guides

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

Related Information

- Intel Quartus Prime Pro Edition User Guide: Getting Started
  Introduces the basic features, files, and design flow of the Intel Quartus Prime Pro Edition software, including managing Intel Quartus Prime Pro Edition projects and IP, initial design planning considerations, and project migration from previous software versions.
- Intel Quartus Prime Pro Edition User Guide: Platform Designer
  Describes creating and optimizing systems using Platform Designer, a system integration tool that simplifies integrating customized IP cores in your project. Platform Designer automatically generates interconnect logic to connect intellectual property (IP) functions and subsystems.
- Intel Quartus Prime Pro Edition User Guide: Design Recommendations
  Describes best design practices for designing FPGAs with the Intel Quartus Prime Pro Edition software. HDL coding styles and synchronous design practices can significantly impact design performance. Following recommended HDL coding styles ensures that Intel Quartus Prime Pro Edition synthesis optimally implements your design in hardware.
- Intel Quartus Prime Pro Edition User Guide: Design Compilation
  Describes set up, running, and optimization for all stages of the Intel Quartus Prime Pro Edition Compiler. The Compiler synthesizes, places, and routes your design before generating a device programming file.
- Intel Quartus Prime Pro Edition User Guide: Design Optimization
  Describes Intel Quartus Prime Pro 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 Pro Edition User Guide: Programmer
  Describes operation of the Intel Quartus Prime Pro 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 Pro Edition User Guide: Block-Based Design
  Describes block-based design flows, also known as modular or hierarchical design flows. These advanced flows enable preservation of design blocks (or logic that comprises a hierarchical design instance) within a project, and reuse of design blocks in other projects.

- Intel Quartus Prime Pro 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 Pro 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 Pro 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 Pro Edition User Guide: Third-party Logic Equivalence Checking Tools
  Describes support for optional logic equivalence checking (LEC) of your design in third-party LEC tools by OneSpin*. Describes how to verify the logic equivalence between compilation netlists.
- Intel Quartus Prime Pro Edition User Guide: Debug Tools
  Describes a portfolio of Intel Quartus Prime Pro 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 Pro Edition User Guide: Timing Analyzer
  Explains basic static timing analysis principals and use of the Intel Quartus Prime Pro 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 Pro Edition User Guide: Power Analysis and Optimization
  Describes the Intel Quartus Prime Pro 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 Pro 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 Interface Planner to prototype interface implementations, plan clocks, and quickly define a legal device floorplan. Use the Pin Planner to visualize, modify, and validate all I/O assignments in a graphical representation of the target device.
- Intel Quartus Prime Pro 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 Pro Edition User Guide: Scripting
  Describes use of Tcl and command line scripts to control the Intel Quartus Prime Pro Edition software and to perform a wide range of functions, such as managing projects, specifying constraints, running compilation or timing analysis, or generating reports.