Intel® Quartus® Prime Pro Edition User Guide

Design Compilation

Updated for Intel® Quartus® Prime Design Suite: **19.1**

Subscribe
Send Feedback

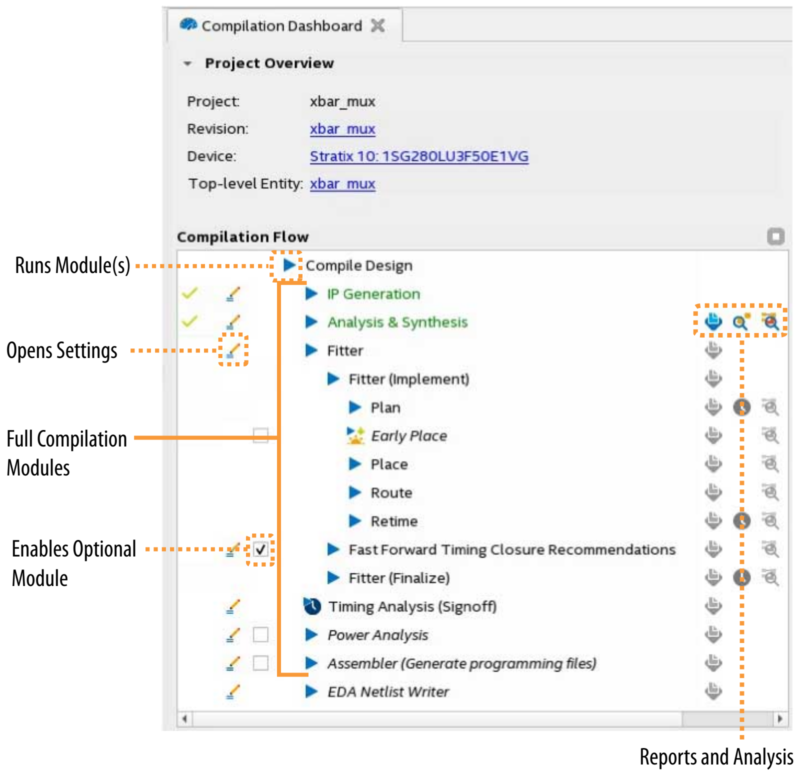
UG-20132 | 2019.07.02
Latest document on the web: PDF | HTML

Contents

1. Design Compilation

The Intel® Quartus® Prime Compiler synthesizes, places, and routes your design before ultimately generating a device programming file. The Compiler supports a variety of high-level, HDL, and schematic design entry methods. The modules of the Compiler include IP Generation, Analysis & Synthesis, Fitter, Timing Analyzer, and Assembler.

Figure 1. Compilation Dashboard



Project Overview

Project: xbar_mux
Revision: xbar_mux
Device: Stratix 10: 1SG280LU3F50E1VG
Top-level Entity: xbar_mux

The Intel Quartus Prime Pro Edition Compiler supports these advanced features:

- Supports Intel Arria® 10, Intel Cyclone® 10 GX, and Intel Stratix® 10 devices.
- Hyper-Aware Design Flow—use Hyper-Retiming and Fast Forward compilation for the highest performance in Intel Stratix 10 devices.
- Partial Reconfiguration—supports dynamic reconfiguration of a portion of the FPGA, while the remaining FPGA continues to function.
- Block-Based Design Flows—enables preservation of design blocks within a project, and reuse of those design blocks in other projects.

1.1. Compilation Overview

The Compiler is modular, allowing you to run only the process that you need. Each Compiler module performs a specific function in the full compilation process. When you run any module, the Compiler runs any prerequisite modules automatically and generates detailed reports at each stage. The Compiler can preserve a "snapshot" of the compilation results after each stage.

Table 1. Compilation Modules

Compilation Process	Description
IP Generation	Identifies the status and version of IP components in the project. Reports outdated IP that require upgrade.
Analysis & Synthesis	Synthesizes, optimizes, minimizes, and maps design logic to device resources. The "synthesized" snapshot preserves the results of this stage. Analysis & Elaboration is a stage of Analysis & Synthesis. This stage checks for design file and project errors.
Fitter (Place & Route)	Assigns the placement and routing of the design to specific device resources, while honoring timing and placement constraints. The Fitter includes the following stages: • Plan—places all periphery elements (such as I/Os and PLLs) and determines a legal clock plan, without core placement or routing. The "planned" snapshot preserves the stage results. • Early Place—places all core elements in an approximate location to facilitate design planning. Finalizes clock planning for Intel Stratix 10 designs. The "early placed" snapshot preserves the stage results. • Place—places all core elements in a legal location. The "placed" snapshot preserves the stage results. • Route—creates all routing between the elements in the design. The "routed" snapshot preserves the stage results. • Retime—moves (retimes) existing registers into Hyper-Registers for fine-grained performance improvement. The "retimed" snapshot preserves the stage results. (1) • Fitter (Finalize)—for Intel Arria 10 and Intel Cyclone 10 GX devices, converts unnecessary tiles to High-Speed or Low-Power. For Intel Stratix 10 devices, performs post-Route fix-up. The "final" snapshot preserves the stage results.
Fast Forward Timing Closure Recommendations	Generates detailed reports that estimate performance gains achievable by making specific RTL modifications.
Timing Analysis	Analyzes and validates the timing performance of all design logic with the Timing Analyzer.
Power Analysis	Optional module that estimates device power consumption. Specify the electrical standard on each I/O cell and the board trace model on each I/O standard in your design.
Assembler	Converts the Fitter's placement and routing assignments into a programming image for the FPGA device.
EDA Netlist Writer	Generates output files for use in other EDA tools.

1.1.1. Using the Compilation Dashboard

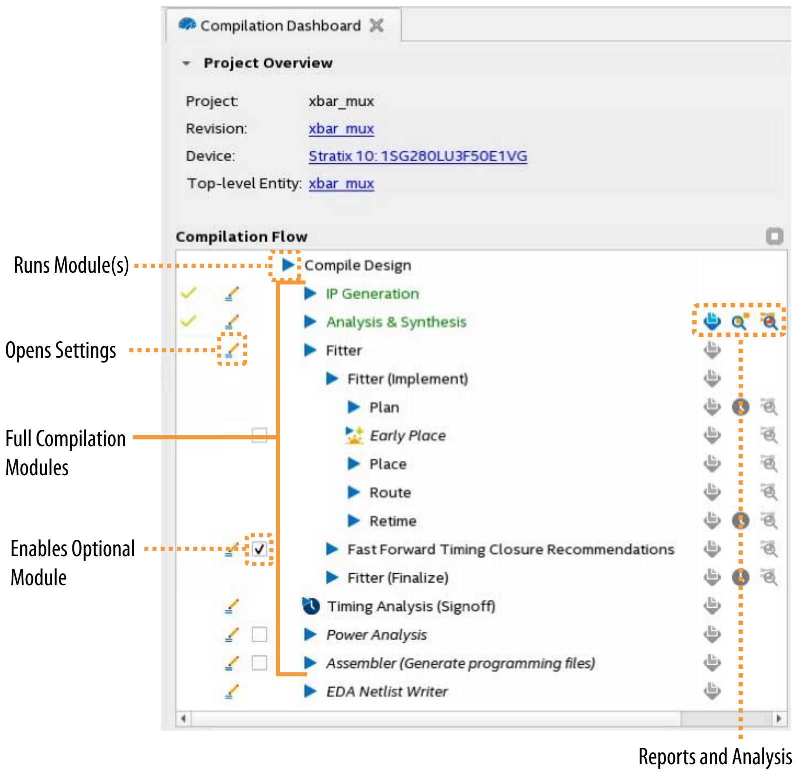
The Compilation Dashboard provides immediate access to settings, controls, and reporting for each stage of the compilation flow.

(1) Retiming and Fast Forward compilation available only for Intel Stratix 10 devices.

The Compilation Dashboard appears by default when you open a project, or you can click **Compilation Dashboard** in the Tasks window to re-open it.

- Click the pencil icon to edit settings for that stage of the compilation flow.
- Click **Compile Design** to run all modules of the Compiler in sequence, or click any individual Compiler module to run compilation through that stage.
- Click the report icon to view reports for that compilation stage.

Figure 2. Compilation Dashboard



Compilation Dashboard

Project Overview

Project:	xbar_mux
Revision:	xbar_mux
Device:	Stratix 10: 1SG280LU3F50E1VG
Top-level Entity:	xbar_mux

Compilation Flow

Compile Design
IP Generation
Analysis & Synthesis
Fitter
Fitter (Implement)
Plan
Early Place
Place
Route
Retime
Fast Forward Timing Closure Recommendations
Fitter (Finalize)
Timing Analysis (Signoff)
Power Analysis
Assembler (Generate programming files)
EDA Netlist Writer

As the Compiler progresses through the flow, the dashboard updates the status of each module, and enables icons that you can click for reports and analysis.

1.1.2. Compilation Flows

The Intel Quartus Prime Pro Edition Compiler supports a variety of flows to help you maximize performance and minimize compilation processing time. The modular Compiler is flexible and efficient, allowing you to run all modules in sequence with a single command, or to run and optimize each stage of compilation separately.

As you develop and optimize your design, run only the Compiler stages that you need, rather than waiting for full compilation. Run full compilation only when your design is complete and you are ready to run all Compiler modules and generate a device programming image.

Table 2. Compilation Flows

Compiler Flow	Function
Early Place Flow	Places all core elements in an approximate location to facilitate design planning. Run **Early Place** to review initial high-level placement of design elements in the Chip Planner or to debug and fine-tune timing constraints. This information is useful to guide your floorplanning decisions.
Fitter (Implement) Flow	Runs the Plan, Early Place, Place, Route, and Retime stages. Run this flow when you are ready to implement placement, routing, and retiming. If successful, you can now perform the Finalize, Timing Analysis, and Assembler stages. [2]
Incremental Optimization Flow	Incremental optimization allows you to stop processing after each Fitter stage, analyze the results, and adjust settings or RTL before proceeding to the next compilation stage. This iterative flow optimizes at each stage, without waiting for full compilation results.
Hyper-Aware Design Flow	Combines automated register retiming (Hyper-Retiming), with implementation of targeted timing closure recommendations (Fast Forward Compilation), to maximize use of Hyper-Registers and drive the highest performance in Intel Stratix 10 devices.
Full Compilation Flow	Launches all Compiler modules in sequence to synthesize, fit, analyze final timing, and generate a device programming file. By default, the Compiler generates and preserves only the synthesized and final snapshots during a full compilation. You can optionally **Enable Intermediate Fitter Snapshots** to preserve the planned, placed, routed, and retimed snapshots.
Partial Reconfiguration	Reconfigures a portion of the FPGA dynamically, while the remaining FPGA design continues to function.
Block-Based Design Flows	Supports preservation and reuse of design blocks in one or more projects. You can reuse synthesized or final design blocks in other projects. Reusable design blocks can include device core or periphery resources.

Related Information

- Incremental Optimization Flow on page 22
- Intel Quartus Prime Pro Edition User Guide: Block-Based Design
- Running Full Compilation on page 35
- Running the Fitter on page 20
- Running the Hyper-Aware Design Flow on page 36
- Intel Quartus Prime Pro Edition User Guide: Partial Reconfiguration

1.1.3. Design Synthesis

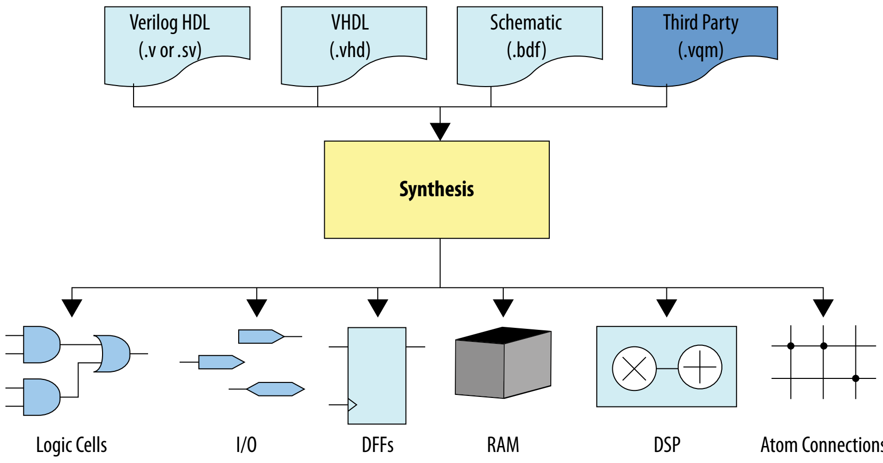
Design synthesis is the process that translates design source files into an atom netlist for mapping to device resources. The Intel Quartus Prime Compiler synthesizes standards-compliant Verilog HDL (`.v`), VHDL (`.vhd`), and SystemVerilog (`.sv`). The Compiler also synthesizes Block Design File (`.bdf`) schematic files, and the Verilog Quartus Mapping (`.vqm`) files generated by other EDA tools.

Synthesis examines the logical completeness and consistency of the design, and checks for boundary connectivity and syntax errors. Synthesis also minimizes and optimizes design logic. For example, synthesis infers D flip flops, latches, and state machines from "behavioral" languages, such as Verilog HDL, VHDL, and SystemVerilog. Synthesis may replace operators, such as + or –, with modules from the Intel Quartus Prime IP library, when advantageous. During synthesis, the Compiler

[2] Retiming and Hyper-Aware design flow only for Intel Stratix 10 devices.

may change or remove user logic and design nodes. Intel Quartus Prime synthesis minimizes gate count, removes redundant logic, and ensures efficient use of device resources.

Figure 3. Design Synthesis



At the end of synthesis, the Compiler generates an atom netlist. Atom refers to the most basic hardware resource in the FPGA device. Atoms include logic cells organized into look-up tables, D flip flops, I/O pins, block memory resources, DSP blocks, and the connections between the atoms. The atom netlist is a database of the atom elements that design synthesis requires to implement the design in silicon.

The Analysis & Synthesis module of the Compiler synthesizes design files and creates one or more project databases for each design partition. You can specify various settings that affect synthesis processing.

The Compiler preserves the results of Analysis & Synthesis in the synthesis snapshot.

1.1.4. Design Place and Route

The Compiler's Fitter module (`quartus_fit`) performs design placement and routing. During place and route, the Fitter determines the best placement and routing of logic in the target FPGA device, while respecting any Fitter settings or constraints that you specify.

By default, the Fitter selects appropriate resources, interconnection paths, and pin locations. If you assign logic to specific device resources, the Fitter attempts to match those requirements, and then fits and optimizes any remaining unconstrained design logic. If the Fitter cannot fit the design in the current target device, the Fitter terminates compilation and issues an error message.

The Intel Quartus Prime Pro Edition Fitter introduces a hybrid placement technique that combines analytical and annealing placement techniques. Analytical placement determines an initial mathematical starting placement. The annealing technique then fine-tunes logic block placement in high resource utilization scenarios.

The Intel Quartus Prime Pro Edition Compiler allows control and optimization of each individual Fitter stage, including the Plan, Early Place, Place, and Route stages. After running a Fitter stage, view detailed report data and analyze the timing of that stage. The Compiler preserves Fitter results of the final snapshot by default.

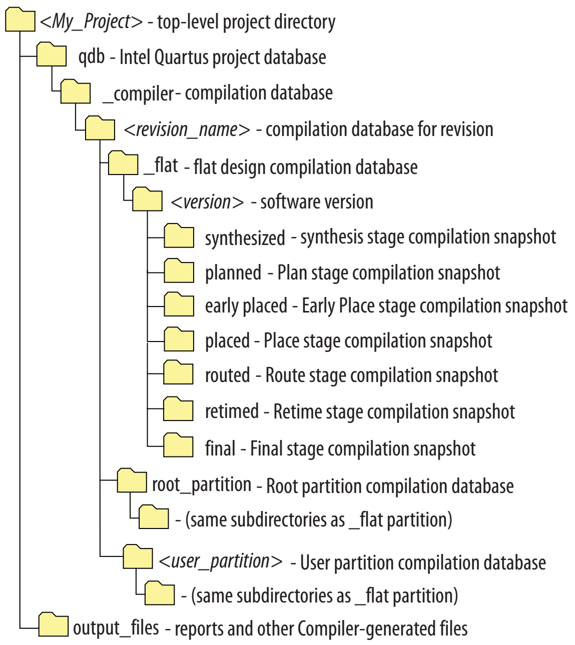
Related Information

- Running the Fitter on page 20
- Viewing Fitter Reports on page 31

1.1.5. Compilation Hierarchy

The Intel Quartus Prime Pro Edition Compiler generates a hierarchical project structure that isolates results of each compilation stage, for each design entity. For example, the `synthesized` directory contains a snapshot of the Analysis & Synthesis stage. If you use design partitions, such as in block-based design, the Compiler also isolates the results for each design partition. The Compiler fully preserves routing and placement within a partition. Changes to other portions of the design hierarchy do not impact the partition. This hierarchical structure allows you to optimize specific design elements without impacting placement and routing in other partitions. The hierarchical project structure also supports distributed work groups and compilation processing across multiple machines.

Figure 4. Hierarchical Project Structure (Intel Stratix 10 Design)


<My_Project> - top-level project directory
qdb - Intel Quartus project database
_compiler- compilation database
<revision_name> - compilation database for revision
_flat - flat design compilation database
<version> - software version
synthesized - synthesis stage compilation snapshot
planned - Plan stage compilation snapshot
early placed - Early Place stage compilation snapshot
placed - Place stage compilation snapshot
routed - Route stage compilation snapshot
retimed - Retime stage compilation snapshot
final - Final stage compilation snapshot
root_partition - Root partition compilation database
- (same subdirectories as _flat partition)
<user_partition> - User partition compilation database
- (same subdirectories as _flat partition)
output_files - reports and other Compiler-generated files

Note: The Compiler only preserves the planned, placed, routed, and retimed snapshots by default during full compilation if you turn on **Enable Intermediate Fitter Snapshots** (**Assignments ➤ Settings ➤ Compiler Settings**)

Related Information

- Exporting Compilation Results on page 50
- Creating a Design Partition on page 52

1.2. Running Synthesis

Run design synthesis as part of a full compilation, or as an independent process. Before running synthesis, specify settings that control synthesis processing. The Messages window dynamically displays processing information, warnings, or errors. Following Analysis & Synthesis processing, the Synthesis report provides detailed information about the synthesis of your project.

To run synthesis:

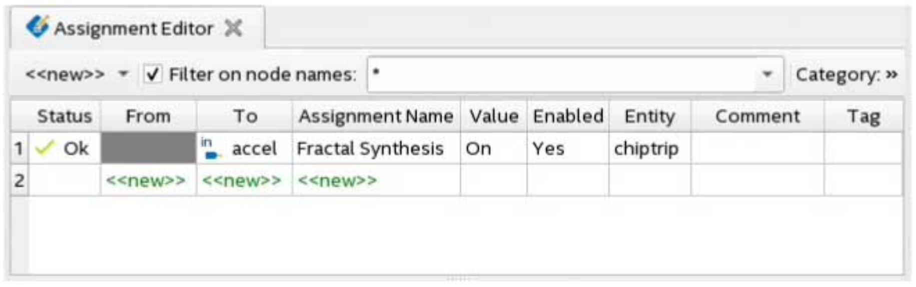
1. Create or open an Intel Quartus Prime project with valid design files for compilation.
2. Before running synthesis, specify any of the following settings and constraints that impact synthesis:
   - To specify options for the synthesis of Verilog HDL input files, click **Assignments ➤ Settings ➤ Verilog HDL Input**.
   - To specify options for the synthesis of VHDL input files, click **Assignments ➤ Settings ➤ VHDL Input**.
   - To specify options that affect compilation processing time, click **Assignments ➤ Settings ➤ Compilation Process Settings**.
   - To specify the Compiler's high-level optimization strategy and other options, click **Assignments ➤ Settings ➤ Compiler Settings**. Specify a **Balanced** strategy, or optimize for **Performance**, **Area**, **Routability**, **Power**, or **Compile Time**. The Compiler targets the optimization goal you specify. Optimization Modes on page 70 describes these options in detail.
   - On the **Compiler Settings** page enable or disable the **Enable Intermediate Fitter Snapshots** option to to generate and preserve snapshots for the Plan, Place, Route, and Retime stages any time you run full compilation. To save compilation time, the Compiler does not generate or preserve these intermediate snapshots by default. However, you must enable this option to use Rapid Recompile.
   - To specify advanced synthesis settings, click **Assignments ➤ Settings ➤ Compiler Settings**, and then click **Advanced Settings (Synthesis)**. Optionally, enable **Timing-Driven Synthesis** to account for timing constraints during synthesis.
3. To enable optional fractal synthesis optimizations for arithmetic-intensive designs that exhaust all DSP resources, follow these steps:
   a. Review the guidelines in Fractal Synthesis Optimization on page 12.

      Caution: Enabling fractal synthesis project-wide causes unnecessary bloat on modules that are not suitable for fractal optimizations.

   b. Click **Assignments ➤ Assignment Editor**.

   c. Select **Fractal Synthesis** for **Assignment Name**, **On** for the **Value**, the arithmetic-intensive entity name for **Entity**, and an instance name in the **To** column. You can enter a wildcard (*) for **To** to assign all instances of the entity.

Figure 5. Fractal Synthesis Assignment in Assignment Editor



	Status	From	To	Assignment Name	Value	Enabled	Entity	Comment	Tag
1	✓ Ok		accel	Fractal Synthesis	On	Yes	chiptrip		
2		<<new>>	<<new>>	<<new>>					

4. To run synthesis, click **Synthesis** on the Compilation Dashboard.

Related Information

- Synthesis Settings Reference on page 70
- Concurrent Analysis During Synthesis or Fitting on page 20

1.2.1. Preserve Registers During Synthesis

Intel Quartus Prime synthesis minimizes gate count, merges redundant logic, and ensures efficient use of device resources. If you need to preserve specific registers through synthesis processing, you can specify any of the following entity-level assignments. Assign the **Preserve Registers in Synthesis** or **Preserve Fan-Out Free Register Node** options to allow Fitter optimization of the preserved registers. **Preserve Registers** restricts Fitter optimization of the preserved registers. Specify synthesis preservation assignments by clicking **Assignments ➤ Assignment Editor**, by modifying the `.qsf` file, or by specifying synthesis attributes in your RTL.

Table 3. Synthesis Preserve Options

Assignment	Description	Allows Fitter Optimization?	Assignment Syntax
Preserve Registers in Synthesis	Prevents removal of registers during synthesis. This settings does not affect retiming or other optimizations in the Fitter.	Yes	• `PRESERVE_REGISTER_SYN_ONLY ON|Off -to <entity>.qsf` • `preserve_syn_only` or `syn_preservesyn_only` (synthesis attributes)
Preserve Fan-Out Free Register Node	Prevents removal of assigned registers without fan-out during synthesis. The PRESERVE_FANOUT_FREE_NODE assignment cannot preserve a fanout-free register that has no fanout inside the Verilog HDL or VHDL module in which you define it. To preserve these fanout-free registers, implement the `noprune` pragma in the source file: `(*noprune*)reg r;` If there are multiple instances of this module, with only some instances requiring preservation of the fanout-free register, set a dummy pragma on the register in the HDL and also	Yes	• `PRESERVE_REGISTER_FANOUT_FREE_NODE ON|Off -to <entity>.qsf` • `no_prune on` (synthesis attribute)

continued...

Assignment	Description	Allows Fitter Optimization?	Assignment Syntax
	set the PRESERVE_FANOUT_FREE_NODE assignment. This dummy pragma allows the register synthesis to implement the assignment. For example, set the following dummy pragma for a register `r` in Verilog HDL: `(*dummy*)reg r;`		
Preserve Registers	Prevents removal and sequential optimization of assigned registers during synthesis. Sequential netlist optimizations can eliminate redundant registers and registers with constant drivers.	No	• `PRESERVE_REGISTER ON|Off -to <entity>.qsf` • `preserve`, `syn_preserve`, or `keep on` (synthesis attributes)

1.2.2. Enabling Timing-Driven Synthesis

Timing-driven synthesis directs the Compiler to account for your timing constraints during synthesis. Timing-driven synthesis runs initial timing analysis to obtain netlist timing information. Synthesis then focuses performance efforts on timing-critical design elements, while optimizing non-timing-critical portions for area.

Timing-driven synthesis preserves timing constraints, and does not perform optimizations that conflict with timing constraints. Timing-driven synthesis may increase the number of required device resources. Specifically, the number of adaptive look-up tables (ALUTs) and registers may increase. The overall area can increase or decrease. Runtime and peak memory use increases slightly.

Intel Quartus Prime Pro Edition runs timing-driven synthesis by default. To enable or disable this option manually, click **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Synthesis)**.

Related Information

- Running Synthesis on page 10
- Synthesis Language Support on page 64

1.2.3. Fractal Synthesis Optimization

You can enable optional fractal synthesis optimizations that are useful for deep-learning accelerators and other high-throughput, arithmetic-intensive designs that exceed all available DSP resources. For such designs, fractal synthesis optimization can achieve 20-45% area reduction.

Caution: Enabling fractal synthesis project-wide causes unnecessary bloat on modules that are not suitable for fractal optimizations.

Table 4. Fractal Synthesis Area Improvement

		Area (LABs)	
Device	**Dot-product**	**Fractal Synthesis ON**	**Fractal Synthesis OFF**
Intel Arria 10	Sum of 16 4x4sm	12	19
	Sum of 16 5x5sm	19	32
	Sum of 16 6x6sm	25	36
	Sum of 16 7x7sm	34	44
	Sum of 16 8x8sm	45	60
Intel Stratix 10	Sum of 16 4x4sm	15	22
	Sum of 16 5x5sm	21	39
	Sum of 16 6x6sm	29	47
	Sum of 16 7x7sm	39	55
	Sum of 16 8x8sm	55	71

Fractal synthesis is a set of synthesis optimizations that use FPGA resources in an optimal way for arithmetic-intensive designs. These synthesis optimizations consist of multiplier regularization and retiming, as well as continuous arithmetic packing. The optimizations target designs with large numbers of low-precision arithmetic operations (such as additions and multiplications).

To use Fractal Synthesis optimizations, identify the basic arithmetic building block that the Fractal Synthesis engine will focus on and mark it using the FRACTAL_SYNTHESIS attribute. You can specify the attribute in one of the following ways:

- In RTL, use `altera_attribute` as follows:

```
(* altera_attribute = "-name FRACTAL_SYNTHESIS ON" *)
```

- In the QSF file, add as an assignment as follows:

```
set_global_assignment -name FRACTAL_SYNTHESIS ON -entity <module name>
```

Note: Fractal synthesis optimization is not recommended for designs without deep-learning accelerators or other high-throughput, arithmetic-intensive functions that exceed all DSP resources. Consider the following factors before enabling fractal synthesis optimization:

Fractal Synthesis Considerations

- Intel Arria 10 and Intel Stratix 10 devices contain thousands of Hard DSP blocks that are perfectly suited for arithmetic operations. If the total amount of arithmetic functions in your design is small, then there is no need to enable **Fractal Synthesis**. In such cases, all the arithmetic functions map directly into DSPs by default. Enable **Fractal Synthesis** only if there are not enough DSP blocks available to implement all arithmetic components. Enable **Fractal Synthesis** only for modules that you do not want the Compiler to map into DSPs.
- In the current version of the Intel Quartus Prime Pro Edition software, fractal synthesis optimizations target low-precision multiplication. Implement high-precision multipliers (where width of every operand exceeds 11 bits) using DSP blocks.
- If you enable **Fractal Synthesis**, the following information message number 20193 may generate during compilation:

  ```
  Applied dense packing to "<entity>". Area: 2 LABs. Logic density: 0.775.
  ```

  This information indicates the effort compiler is making to pack computational logic into smaller number of LABs. If the design is already highly utilized, this effort might be skipped.

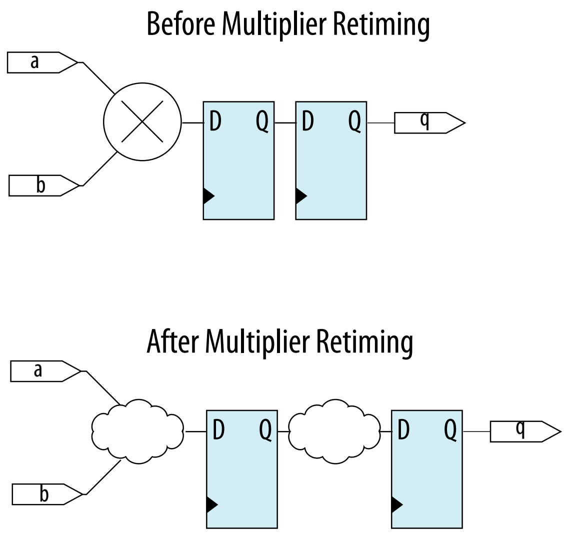
  — Verify that the Area the message reports does not exceed 100 LABs. If the Area exceeds 100 LABs, divide fractal synthesis blocks to sub-blocks, and then assign the fractal synthesis optimizations to the sub-blocks independently.

  — Verify that the Logic density the message reports is greater than 0.75. If the logic density is less than 0.75, disable **Fractal Synthesis** for this entity because standard synthesis typically achieves better density.

Multiplier Regularization and Retiming

Multiplier regularization and retiming performs inference of highly optimized soft multiplier implementations. The Compiler may apply backward retiming to two or more pipeline stages if required. When you enable fractal synthesis, the Compiler applies multiplier regularization and retiming to signed and unsigned multipliers.

Figure 6. Multiplier Retiming



Note:
- Multiplier regularization uses only logic resources, and does not use DSP blocks.
- Multiplier regularization and retiming is applied to both signed and unsigned multipliers in modules where the FRACTAL_SYNTHESIS QSF assignment is set.

Multiplier Regularization Example

Simple Unsigned Dot-Product Design

The following simple, unsigned dot-product design example contains multiplication operators with 5-bit operands. These short multipliers are perfect candidates for multiplier regularization.

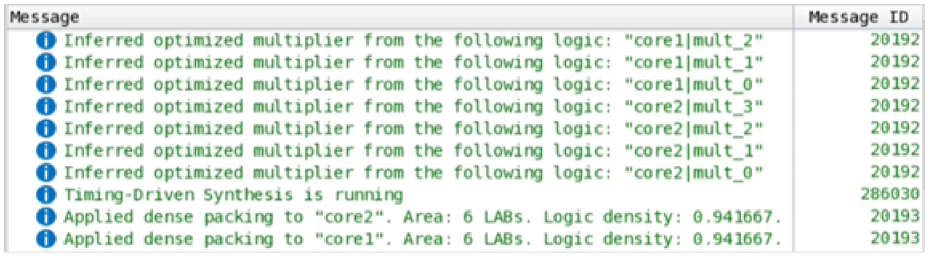
```
(* altera_attribute = "-name FRACTAL_SYNTHESIS ON" *)
module dot_product(
   input clk,
   input [4:0] a, b, c, d, e, f, g, h,
   output reg [11:0] out
);
reg [9:0] ab, cd, ef, gh;
reg [10:0] ab_cd, ef_gh;

always @(posedge clk)
begin
    ab <= a * b;
    cd <= c * d;
    ef <= e * f;
    gh <= g * h;
    ab_cd <= ab + cd;
    ef_gh <= ef + gh;
    out <= ab_cd + ef_gh;
end
endmodule

module top(
   input clk,
   input [4:0] a1, b1, c1, d1, e1, f1, g1, h1,
   input [4:0] a2, b2, c2, d2, e2, f2, g2, h2,
   output [11:0] out1, out2
);
dot_product core1(.clk(clk), .a(a1), .b(b1), .c(c1), .d(d1),
    .e(e1), .f(f1), .g(g1), .h(h1), .out(out1));
dot_product core2(.clk(clk), .a(a2), .b(b2), .c(c2), .d(d2),
    .e(e2), .f(f2), .g(g2), .h(h2), .out(out2));
endmodule
```

Quartus Synthesis prints the following messages on the console:

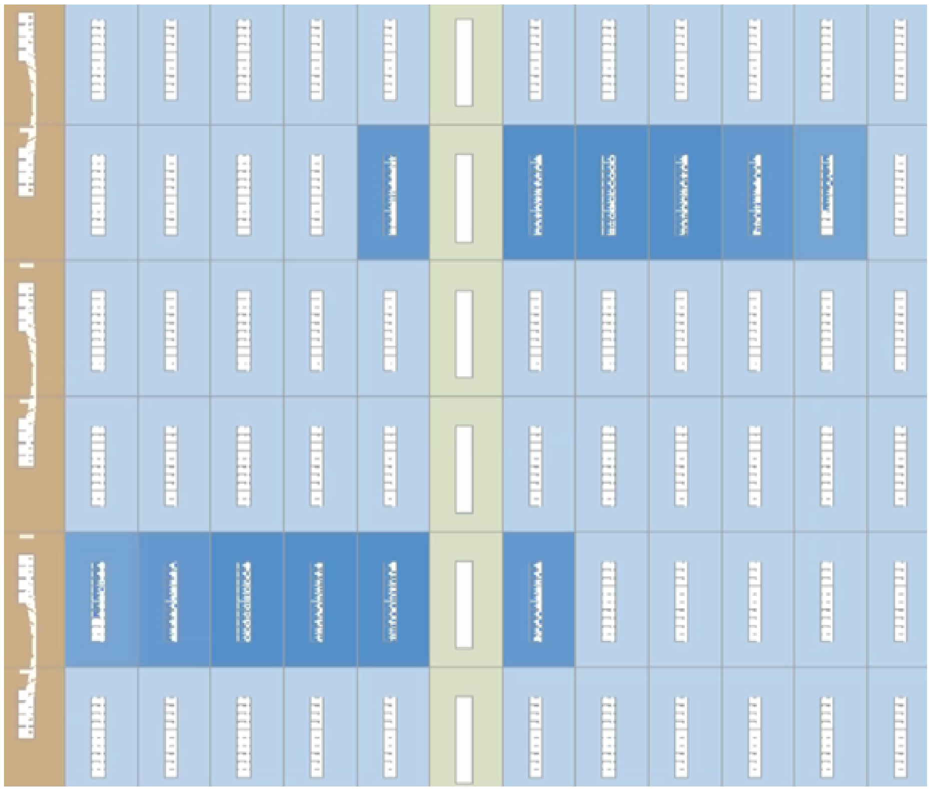
Figure 7. Console Messages



```
Message                                                                   Message ID
Inferred optimized multiplier from the following logic: "core1|mult_2"         20192
Inferred optimized multiplier from the following logic: "core1|mult_1"         20192
Inferred optimized multiplier from the following logic: "core1|mult_0"         20192
Inferred optimized multiplier from the following logic: "core2|mult_3"         20192
Inferred optimized multiplier from the following logic: "core2|mult_2"         20192
Inferred optimized multiplier from the following logic: "core2|mult_1"         20192
Inferred optimized multiplier from the following logic: "core2|mult_0"         20192
Timing-Driven Synthesis is running                                            286030
Applied dense packing to "core2". Area: 6 LABs. Logic density: 0.941667.       20193
Applied dense packing to "core1". Area: 6 LABs. Logic density: 0.941667.       20193
```

In the Chip Planner, this design can be observed having two unsigned dot-product cores independently optimized and placed, and LAB resources almost 100% optimized, as illustrated in the following image:

Figure 8. Design Placement



Signed Dot-Product Design

Signed dot-products are common for deep-learning applications. The following demonstrates an example of a signed dot-product:

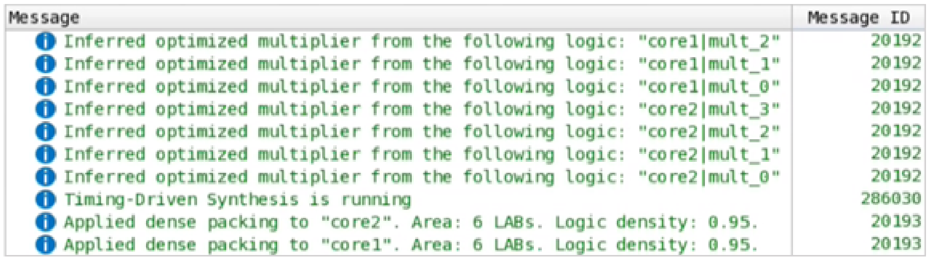
```
(* altera_attribute = "-name FRACTAL_SYNTHESIS ON" *)
module dot_product(
   input signed clk,
   input signed [4:0] a, b, c, d, e, f, g, h,
   output reg signed [11:0] out
);
reg signed [9:0] ab, cd, ef, gh;
reg signed [10:0] ab_cd, ef_gh;

always @(posedge clk)
begin
    ab <= a * b;
    cd <= c * d;
    ef <= e * f;
    gh <= g * h;
    ab_cd <= ab + cd;
    ef_gh <= ef + gh;
    out <= ab_cd + ef_gh;
end
endmodule

module top(
   input clk,
   input signed [4:0] a1, b1, c1, d1, e1, f1, g1, h1,
   input signed [4:0] a2, b2, c2, d2, e2, f2, g2, h2,
   output signed [11:0] out1, out2
);
dot_product core1(.clk(clk), .a(a1), .b(b1), .c(c1), .d(d1),
    .e(e1), .f(f1), .g(g1), .h(h1), .out(out1));
dot_product core2(.clk(clk), .a(a2), .b(b2), .c(c2), .d(d2),
    .e(e2), .f(f2), .g(g2), .h(h2), .out(out2));
endmodule
```

Quartus Synthesis prints the following messages on the console:

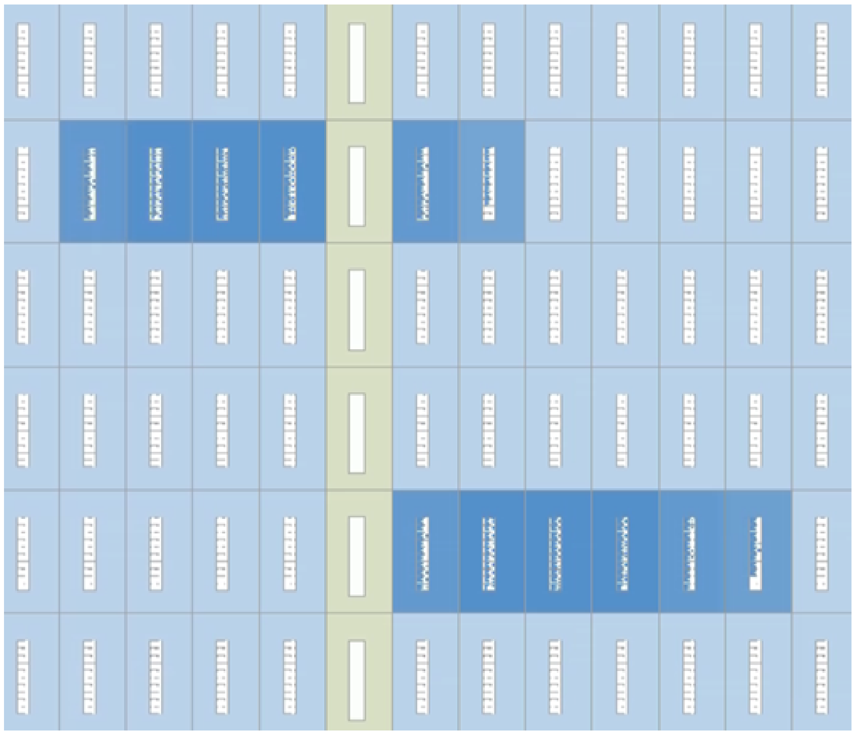
Figure 9. Console Messages



```
Message                                                                     Message ID
Inferred optimized multiplier from the following logic: "core1|mult_2"          20192
Inferred optimized multiplier from the following logic: "core1|mult_1"          20192
Inferred optimized multiplier from the following logic: "core1|mult_0"          20192
Inferred optimized multiplier from the following logic: "core2|mult_3"          20192
Inferred optimized multiplier from the following logic: "core2|mult_2"          20192
Inferred optimized multiplier from the following logic: "core2|mult_1"          20192
Inferred optimized multiplier from the following logic: "core2|mult_0"          20192
Timing-Driven Synthesis is running                                             286030
Applied dense packing to "core2". Area: 6 LABs. Logic density: 0.95.            20193
Applied dense packing to "core1". Area: 6 LABs. Logic density: 0.95.            20193
```

In the Chip Planner, this design can be observed having two signed dot-product cores independently optimized and placed:

Figure 10. Design Placement

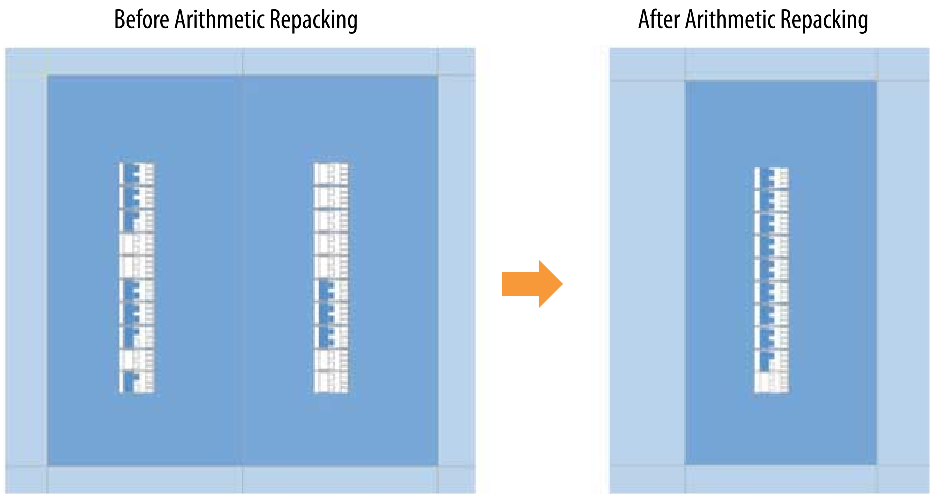


Continuous Arithmetic Packing

Continuous arithmetic packing re-synthesizes arithmetic gates into logic blocks optimally sized to fit into Intel FPGA LABs. This optimization allows up to 100% utilization of LAB resources for the arithmetic blocks.

When you enable fractal synthesis, the Compiler applies this optimization to all carry chains and two-input logic gates. This optimization can pack adder trees, multipliers, and any other arithmetic-related logic.

Figure 11. Continuous Arithmetic Packing

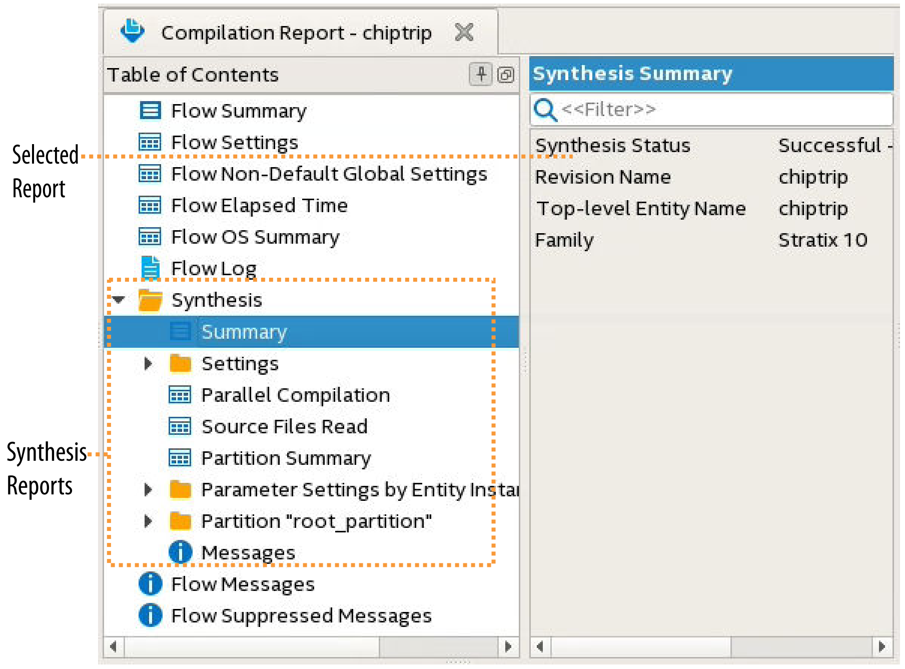


Note that continuous arithmetic packing works independently of multiplier regularization. So, if you are using a multiplier that is not regularized (such as writing your own multiplier) then continuous arithmetic packing can still operate.

1.2.4. Synthesis Reports

The Compilation Report window opens automatically during compilation processing. The Report window displays detailed synthesis results for each partition in the current project revision.

Figure 12. Synthesis Reports



Synthesis Status	Successful -
Revision Name	chiptrip
Top-level Entity Name	chiptrip
Family	Stratix 10

Table 5. Synthesis Reports (Design Dependent)

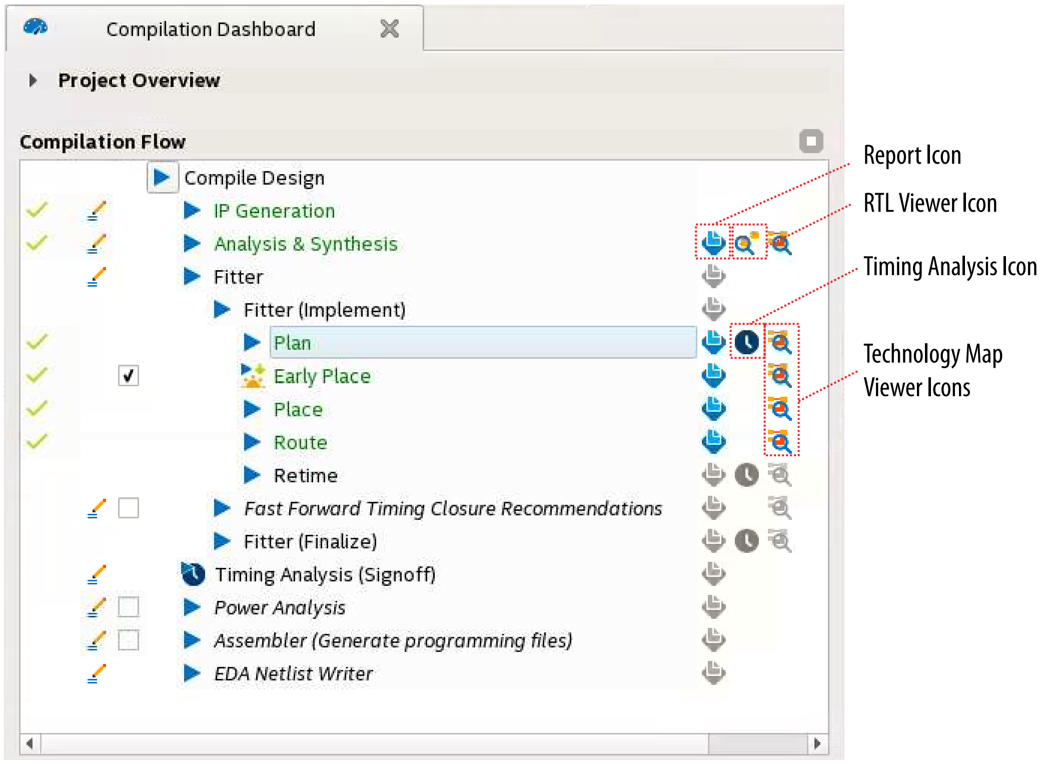
Generated Report	Description
Summary	Shows summary information about synthesis, such as the status, date, software version, entity name, device family, timing model status, and various types of logic utilization.
Synthesis Settings	Lists the values of all synthesis settings during design processing.
Parallel Compilation	Lists specifications for any use of parallel processing during synthesis.
Resource Utilization By Entity	Lists the quantity of all types of logic usage for each entity in design synthesis.
Multiplexer Restructuring Statistics	Provides statistics for the amount of multiplexer restructuring that synthesis performs.
IP Cores Summary	Lists details about each IP core instance in design synthesis. Details include IP core name, vendor, version, license type, entity instance, and IP include file.
Synthesis Source Files Read	Lists details about all source files in design synthesis. Details include file path, file type, and any library information.
Resource Usage Summary for Partition	Lists the quantity of all types of logic usage for each design partition in design synthesis.
RAM Summary for Partition	Lists RAM usage details for each design partition in design synthesis. Details include the name, type, mode, and density.
Register Statistics	Lists the number of registers using various types of global signals.
Synthesis Messages	Lists all information, warning, and error messages that report conditions observed during the Analysis & Synthesis process.

1.2.5. Concurrent Analysis During Synthesis or Fitting

If you run Analysis & Synthesis, or the Fitter, you can access results while downstream Fitter stages are still running. Once the **Concurrent Analysis** icons become active in the dashboard, you can view the analysis without interrupting compilation.

During Analysis & Synthesis, you can click the **Concurrent Analysis** icons on the Dashboard to view reports, the RTL Viewer, or the Technology Map Viewer. While the Fitter is processing, you can analyze timing during the stages displaying the **Timing Analyzer** icon, and view Technology Map Viewer snapshots during Fitter stages. You should not modify timing constraints during concurrent analysis, because it affects the results of the underlying compile. However, you can halt a compile at any time, modify the `.sdc` constraints in your source file, and then click the **Timing Analyzer** icon to restart analysis with the modified constraints.

Figure 13. Concurrent Analysis Options



1.3. Running the Fitter

The Compiler's Fitter module performs all stages of design place and route, including the Plan, Early Place, Place, Route, and Retime stages. Run all stages of the Fitter as part of a full design compilation, or run any Fitter stage independently after design synthesis. Before running the Fitter, you specify settings that impact Fitter processing.

1. Specify initial Fitter constraints:
   a. To assign device I/O pins, click **Assignments ➤ Pin Planner**.
   b. To assign device periphery, clocks, and I/O interfaces, click **Tools ➤ Interface Planner**.

c. To constrain logic placement regions, click **Tools ➤ Chip Planner**.

d. To specify Fitter optimization goals, click **Assignments ➤ Settings ➤ Compiler Settings**. Optimization Modes on page 70 describes these options in detail

e. To fine-tune place and route with advanced Fitter options, click **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Fitter)**

2. To run one or more stages of the Fitter, click any of the following commands on the Compilation Dashboard:
   - To run all Fitter stages in sequence, click **Fitter**.
   - To run only device periphery placement and routing, click **Plan**.
   - To run only early placement, click **Early Place**.
   - To run only logic placement, click **Place**.
   - To run only logic routing, click **Route**.
   - To run only retiming of ALM registers into Hyper-Registers, click **Retime**.(3)
   - To run the Implement flow (runs Plan, Place, Route, and Retime stages), click **Fitter (Implement)**.
   - To run the Finalize flow (runs Plan, Early Place, Place, Route, Retime, and Finalize stages), click **Fitter (Finalize)**.

Related Information

- Fitter Settings Reference on page 77
- Step 2: Review Retiming Results on page 40

1.3.1. Fitter Stage Commands

Launch Fitter processes from the Processing menu or Compilation Dashboard.

Table 6. Fitter Stage Commands

Command	Description
Fitter (Implement)	Runs the Plan, Early Place, Place, Route, and Retime stages.
Start Fitter (Plan)	Loads synthesized periphery placement data and constraints, and assigns periphery elements to device I/O resources. After this stage, you can run post-Plan timing analysis to verify timing constraints, and validate cross-clock timing windows. View the placement and properties of periphery (I/O) and perform clock planning for Intel Arria 10 and Intel Cyclone 10 GX designs. This command creates the planned snapshot.
Start Fitter (Early Place)	Places all core elements in an approximate location to facilitate design planning. After this stage, the Chip Planner displays initial high-level placement of design elements. The Compilation reports identifies high fan-out signals that increase placement complexity. Use this information to guide your floorplanning decisions. For Intel Stratix 10 designs, you can also do early clock planning after this stage. This command creates the early placed snapshot. Early Place does not run during the full compilation flow by default, but you can enable by default or run directly from the Compilation Dashboard.
Start Fitter (Place)	Places all core elements in a legal location. This command creates the placed snapshot.

continued...

(3) Retime available for Intel Stratix 10 devices only.

Command	Description
Start Fitter (Route)	Creates all routing between the elements in the design. After this stage, validate delay chain settings and analyze routing resources. Perform detailed setup and hold timing closure in the Timing Analyzer and view routing congestion via the Chip Planner. This command creates the routed snapshot.
Start Fitter (Retime)	Performs register retiming and moves existing registers into Hyper-Registers to increase performance by removing retiming restrictions and eliminating critical paths. The Compiler may report hold violations for short paths following the Retime stage. The Fitter identifies and corrects the short paths with hold violations during the Fitter (Finalize) stage by adding routing wire along the paths. This command creates the retimed snapshot.
Start Fitter (Finalize)	Performs post-routing optimization on the design. This stage converts unneeded tiles from High Speed to Low Power. This command creates the final snapshot. For Intel Stratix 10 designs, the Fitter also runs post-route fix-up to correct any short path hold violations remaining from retiming.

Related Information

Concurrent Analysis During Synthesis or Fitting on page 20

1.3.2. Incremental Optimization Flow

Intel Quartus Prime Pro Edition supports incremental optimization at each stage of design compilation. In incremental optimization, you run and optimize each compilation stage independently before running the next compilation module in sequence. The Compiler preserves the results of each stage as a snapshot for analysis. When you make changes to your design or constraints, the Compiler only runs stages impacted by the change. Following synthesis or any Fitter stage, view results and perform timing analysis. Modify design RTL or Compiler settings, as needed. Then, re-run synthesis or the Fitter and evaluate the results of these changes. Repeat this process until the module performance meets requirements. This flow maximizes the results at each stage, without waiting for full compilation results.

Figure 14. Incremental Optimization Flow

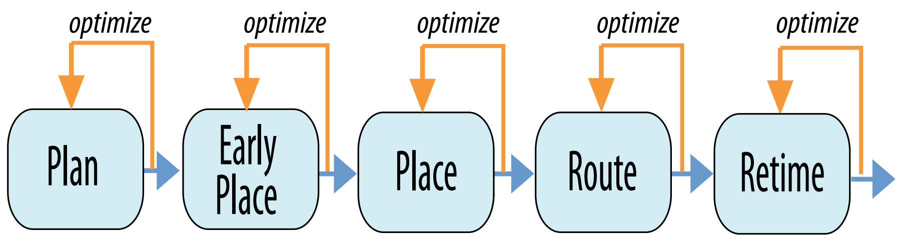


Table 7. Incremental Optimization at Fitter Stages

Fitter Stage	Incremental Optimization
Plan	After this stage, you can run post-Plan timing analysis to verify timing constraints, and validate cross-clock timing windows. View the placement and properties of periphery (I/O) and perform clock planning for Intel Arria 10 and Intel Cyclone 10 GX designs.
Early Place	After this stage, the Chip Planner can display initial high-level placement of design elements. Use this information to guide your floorplanning decisions. For Intel Stratix 10 designs, you can also review and modify clock assignments after performing this stage.

continued...

Fitter Stage	Incremental Optimization
Place	After this stage, validate resource and logic utilization in the Compilation Reports, and review placement of design elements in the Chip Planner.
Route	After this stage, perform detailed setup and hold timing closure in the Timing Analyzer, and view routing congestion via the Chip Planner.
Retime	After this stage, review the Retiming results in the Fitter report and correct any restrictions limiting further retiming optimization.(4)

1.3.2.1. Early Place Flow

Early Place begins assigning core logic to device resources. Run **Early Place** to quickly view the effect of iterative floorplanning changes, without waiting for full placement or full compilation. The Compiler preserves a snapshot of the Early Place results. Following Early Place, click the Timing Analyzer icon to validate your `.sdc` constraints. Do not use the Early Place timing results to compare with Final timing results, as timing between early snapshots and the final snapshot are not well correlated.

Early Place runs automatically during Fitter processing if you enable the **Early Place** stage on the compilation dashboard, or by enabling **Settings ➤ Compiler Settings ➤ Fitter Settings (Advanced) ➤ Run Early Place During Compilation**.

1.3.2.2. Running late_place After Early Place

After running the Early Place stage, you can run `late_place`, rather than the full Place stage, to reduce total compilation time. `late_place` skips the placements that Early Place makes. The Place stage includes the Early Place and `late_place` stages. The Intel Quartus Prime GUI does not support the `late_place` option. The `late_place` option is only available at the command line, after running the Early Place stage from the GUI or from the command line. Running `late_place` generates the placed snapshot. Access command-line help to display details about the `late_place` argument.

Note: Type `quartus_fit –help=late_place` for command-line help on this argument.

To run `late_place` after Early Place:

1. To run the Early Place stage and generate the Early Place snapshot, perform one of the following:
   - To run Early Place in the GUI, click **Early Place** on the Compilation Dashboard. The Compiler runs any required prerequisite stages.
   - To run Early Place (and prerequisite stages) at the command-line, run the following commands. The "-" character equals double hyphens:

   ```
   quartus_ipgenerate <design_name>
   quartus_syn <design name>
   quartus_fit -plan <design name>
   quartus_fit -early_place <design name>
   ```

   Note: You must generate the early placed snapshot before running `late_place`, or the Fitter reports an error.

(4) Retiming available only for Intel Stratix 10 devices.

2. View Early Place results in the Early Placed Fitter reports of the Compilation Report, and in the Chip Planner (**Tools ➤ Chip Planner**).
3. When satisfied with the Early Place results, type one of the following commands to continue to the `late_place` stage and beyond. View `late_place` results and processing messages in the <*design name*>`.fit.place.rpt` file.

   -
     ```
     quartus_fit –late_place <design name>
     (runs late_place)
     ```

   -
     ```
     quartus_fit –late_place –route <design name>
     (runs late_place and route)
     ```

   -
     ```
     quartus_fit –late_place –route –finalize <design name>
     (runs late_place and finalize)
     ```

1.3.3. Analyzing Fitter Snapshots

Analyze the results of Fitter stages to evaluate your design before running the next stage, or before running a full compilation. Use this technique to isolate potential problems and reduce the overall time you spend running design compilation. The following topics describe typical use cases for analyzing Fitter snapshots.

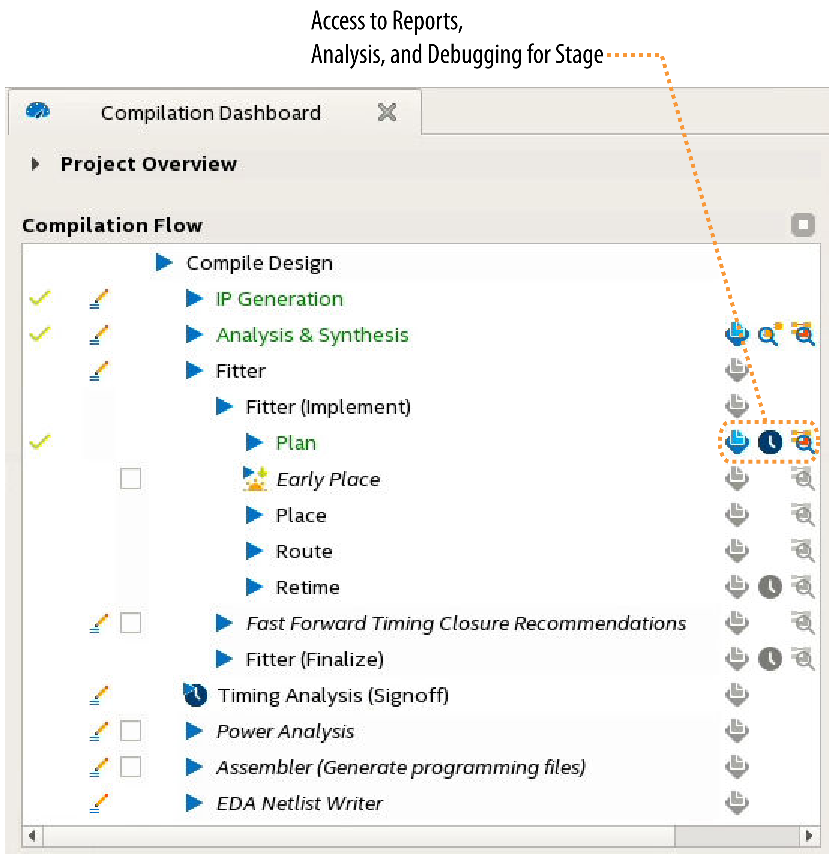
Note: The Compiler saves the planned, placed, routed, and retimed snapshots by default during full compilation only if you turn on **Enable Intermediate Fitter Snapshots** (**Assignments ➤ Settings ➤ Compiler Settings**). You can also run any intermediate Fitter stage independently to generate the snapshot for that stage.

1.3.3.1. Validating SDC Constraints after the Plan Stage

The Fitter's Plan stage performs initial validation of your project's `.sdc` constraints. The Compiler generates messages during the plan stage that warn you about any possible invalid `.sdc` constraints. Stop compilation following the Plan stage to validate and make any necessary changes to `.sdc` constraints, before moving on to the next Fitter stage. To validate `.sdc` constraints after the Plan stage, follow these steps:

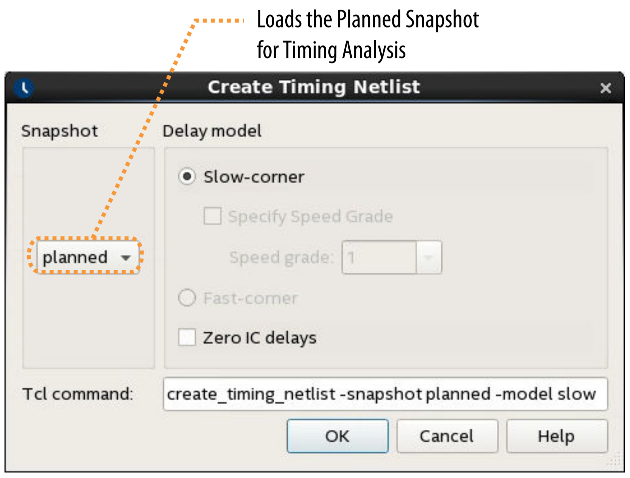
1. To run the Fitter's Plan stage, click **Plan** on the Compilation Dashboard. The Compiler automatically runs prerequisite compilation stages, if necessary.
2. On the Compilation Dashboard, click the **Timing Analyzer** icon adjacent to the Fitter stage. The **Create Timing Netlist** dialog box appears and loads the corresponding stage snapshot.

Figure 15. Plan Stage Timing Analyzer Icon in Compilation Dashboard



3. In the **Create Timing Netlist** dialog box, click **OK**. The planned database loads in the Timing Analyzer.

Figure 16. Planned Snapshot in Create Timing Netlist Dialog Box



Snapshot: planned

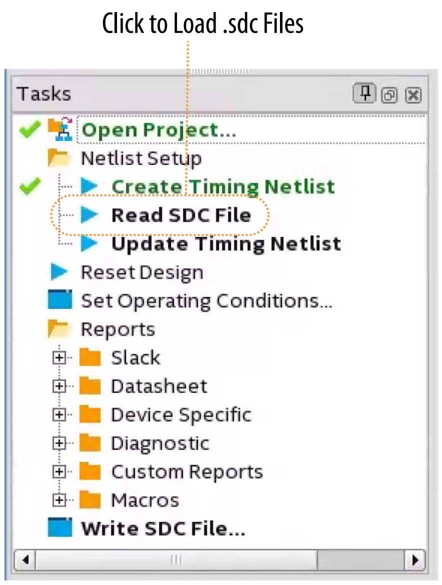
Delay model: (● Slow-corner) ☐ Specify Speed Grade, Speed grade: 1, ○ Fast-corner, ☐ Zero IC delays

Tcl command:

```
create_timing_netlist -snapshot planned -model slow
```

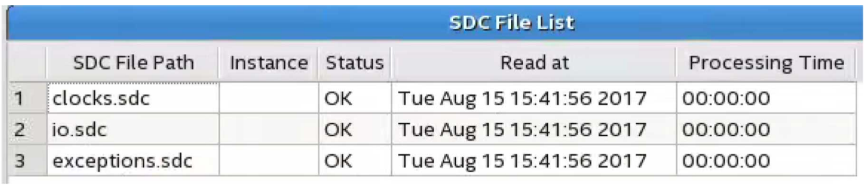
4. On the **Tasks** pane, click **Read SDC File**. The Timing Analyzer reads and processes any `.sdc` files. For multiple `.sdc` files, the report also includes the `.sdc` processing sequence.

Figure 17. Read SDC File Command



5. To report the `.sdc` constraints that apply to the project, click **Report SDC** under the **Diagnostic** folder, in the **Tasks** pane.

Figure 18. SDC File List Report



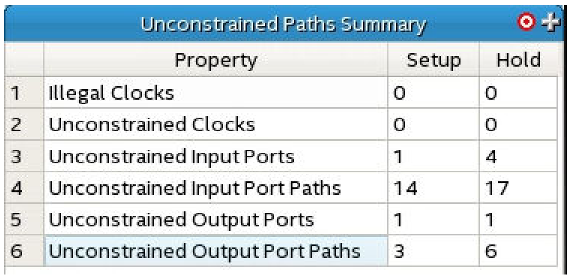
SDC File List

	SDC File Path	Instance	Status	Read at	Processing Time
1	clocks.sdc		OK	Tue Aug 15 15:41:56 2017	00:00:00
2	io.sdc		OK	Tue Aug 15 15:41:56 2017	00:00:00
3	exceptions.sdc		OK	Tue Aug 15 15:41:56 2017	00:00:00

6. Conversely, to report the constraints in the `.sdc` files that the Timing Analyzer ignores, click **Report Ignored Constraints** under the **Diagnostic** folder, in the **Tasks** pane.

7. To report all paths in your design that have no constraints, click **Report Unconstrained Paths** under the **Diagnostic** folder, in the **Tasks** pane.

Figure 19. Unconstrained Paths Summary

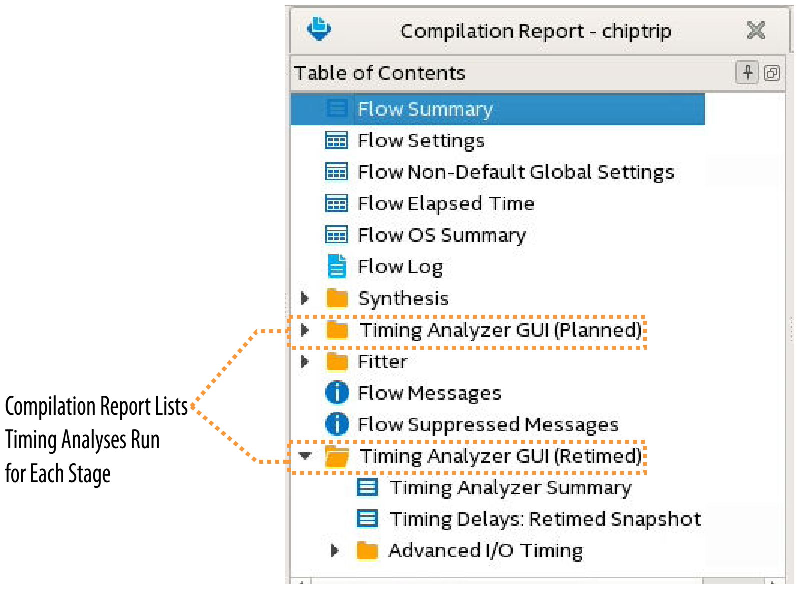


Unconstrained Paths Summary

	Property	Setup	Hold
1	Illegal Clocks	0	0
2	Unconstrained Clocks	0	0
3	Unconstrained Input Ports	1	4
4	Unconstrained Input Port Paths	14	17
5	Unconstrained Output Ports	1	1
6	Unconstrained Output Port Paths	3	6

The Compilation Report displays the Timing Analysis that you run for each stage.

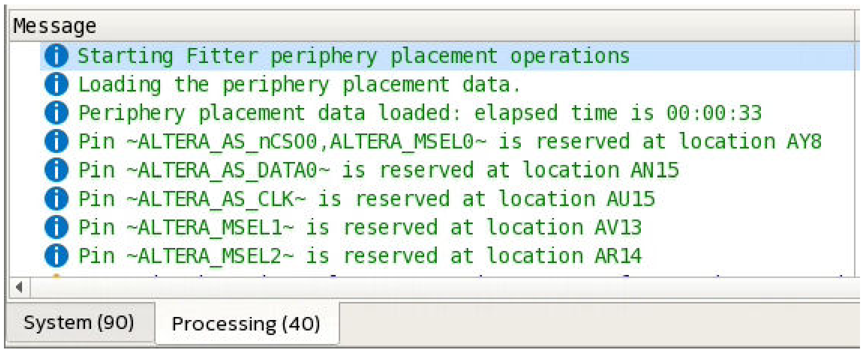
Figure 20. Plan and Retime Stage Timing Analysis Reports in Compilation Report


Compilation Report - chiptrip
Table of Contents
Flow Summary
Flow Settings
Flow Non-Default Global Settings
Flow Elapsed Time
Flow OS Summary
Flow Log
Synthesis
Timing Analyzer GUI (Planned)
Fitter
Flow Messages
Flow Suppressed Messages
Timing Analyzer GUI (Retimed)
Timing Analyzer Summary
Timing Delays: Retimed Snapshot
Advanced I/O Timing
Compilation Report Lists Timing Analyses Run for Each Stage

1.3.3.2. Validating Periphery (I/O) after the Plan Stage

The Compiler begins periphery placement during the Plan stage, and reports data about periphery elements, such as I/O pins and PLLs. After the Plan stage, view the Compilation Report to evaluate the placement of periphery elements before proceeding to the next compilation stage.

Figure 21. Plan Stage Periphery Placement Message



```
Message
Starting Fitter periphery placement operations
Loading the periphery placement data.
Periphery placement data loaded: elapsed time is 00:00:33
Pin ~ALTERA_AS_nCSO0,ALTERA_MSEL0~ is reserved at location AY8
Pin ~ALTERA_AS_DATA0~ is reserved at location AN15
Pin ~ALTERA_AS_CLK~ is reserved at location AU15
Pin ~ALTERA_MSEL1~ is reserved at location AV13
Pin ~ALTERA_MSEL2~ is reserved at location AR14
```

1. In the Compilation Dashboard, click the **Plan** stage.
2. In the Compilation Report, under the **Plan Stage** folder, click the **Input Pins**, **Output Pins**, **I/O Bank Usage**, **PLL Usage Summary**, or other reports. Verify attributes of the I/O pins, such as the physical pin location, I/O standards, and PLL placement.

Figure 22. Input Pins Report

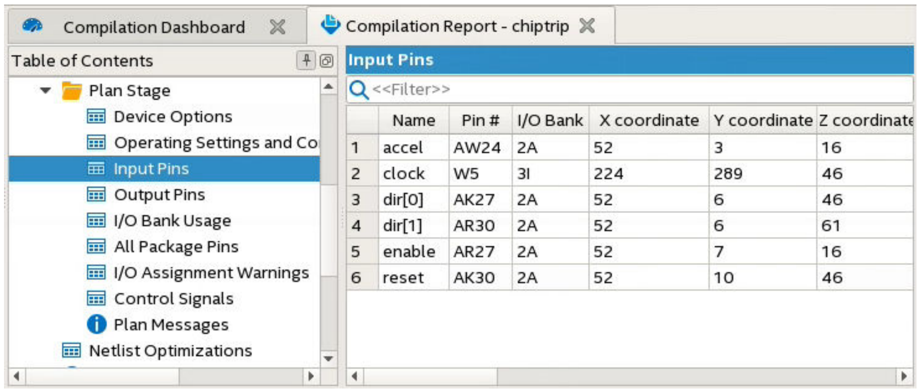

Compilation Dashboard | Compilation Report - chiptrip

Table of Contents
- Plan Stage
  - Device Options
  - Operating Settings and Co
  - Input Pins
  - Output Pins
  - I/O Bank Usage
  - All Package Pins
  - I/O Assignment Warnings
  - Control Signals
  - Plan Messages
- Netlist Optimizations

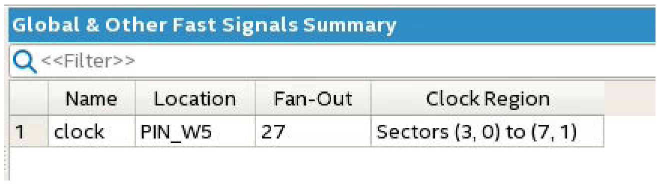
Input Pins

<<Filter>>

	Name	Pin #	I/O Bank	X coordinate	Y coordinate	Z coordinate
1	accel	AW24	2A	52	3	16
2	clock	W5	3I	224	289	46
3	dir[0]	AK27	2A	52	6	46
4	dir[1]	AR30	2A	52	6	61
5	enable	AR27	2A	52	7	16
6	reset	AK30	2A	52	10	46

3. For Intel Arria 10 and Intel Cyclone 10 GX designs, click **Global & Other Fast Signals Summary** report to verify which clocks the Compiler promotes to global clocks. Clock planning occurs after the Early Place stage for Intel Stratix 10 designs.

Figure 23. Global & Other Fast Signals Report Shows Clock Promotion (Intel Arria 10 and Intel Cyclone 10 GX FPGAs)


Global & Other Fast Signals Summary

<<Filter>>

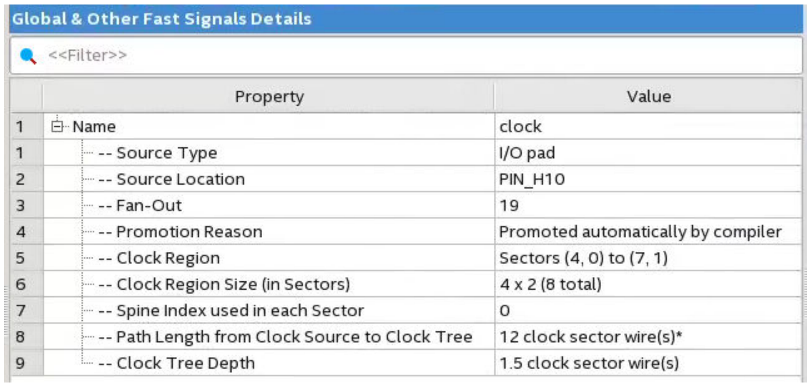
	Name	Location	Fan-Out	Clock Region
1	clock	PIN_W5	27	Sectors (3, 0) to (7, 1)

1.3.3.3. Clock Planning after Early Place (Intel Stratix 10 only)

Intel Stratix 10 devices support clock planning after the Early Place stage, rather than after the Plan stage. After running Early Place, view the Global & Other Fast Signals report to view details and plan the clocks in your project. To view clock details after Early Place, follow these steps:

1. In the Compilation Dashboard, click the **Early Place** stage.
2. In the Compilation Report, under the **Early Place Stage** folder, click the **Global & Other Fast Signals Details** or **Global & Other Fast Signals Summary** report.

Figure 24. Global & Other Fast Signal Details Report


Global & Other Fast Signals Details

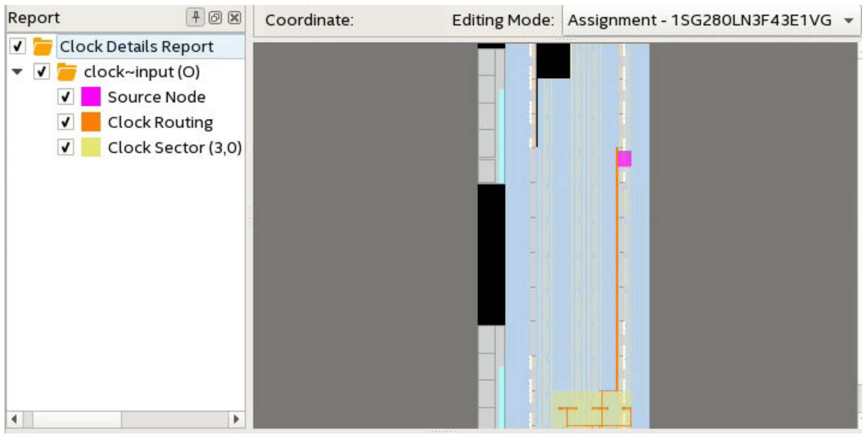
<<Filter>>

	Property	Value
1	Name	clock
1	-- Source Type	I/O pad
2	-- Source Location	PIN_H10
3	-- Fan-Out	19
4	-- Promotion Reason	Promoted automatically by compiler
5	-- Clock Region	Sectors (4, 0) to (7, 1)
6	-- Clock Region Size (in Sectors)	4 x 2 (8 total)
7	-- Spine Index used in each Sector	0
8	-- Path Length from Clock Source to Clock Tree	12 clock sector wire(s)*
9	-- Clock Tree Depth	1.5 clock sector wire(s)

The report provides clock tree path length and depth. The shortest path length from clock source to clock tree, and the smallest clock tree depth, results in the best clock performance.

3. To visualize the clock path length and clock tree depth, click **Tools ➤ Chip Planner**.
4. In the Chip Planner **Tasks** pane, click **Report Clock Details** under the **Clock Reports** folder.
5. In the **Report Clock Details** dialog box, click **OK**. The **Report** pane lists all the clocks in the design.
6. In the Report pane, select one or more clocks to highlight the clock elements in Chip Planner.

Figure 25. Visualizing Clocks in Chip Planner



After running the Early Place stage, you can run `late_place`, rather than the full Place stage, to reduce total compilation time.

Related Information

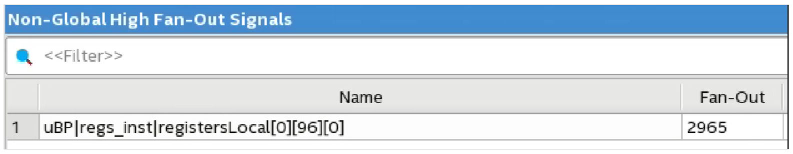
Use CLOCK_REGION to Optimize Clock Constraints, Design Recommendations User Guide

1.3.3.4. Identifying High Fan-Out Signals after Early Place

High fan-out signals increase placement difficulty. After Early Place, identify and consider moving high fan-out signals to global resources.

1. In the Compilation Dashboard, click the **Early Place** stage.
2. In the Compilation Report, under the **Early Place Stage** folder, click the **Non-Global High Fan-Out Signals** report. The report lists the number of fan-outs for each signal.

Figure 26. Non-Global High Fan-Out Signals Report



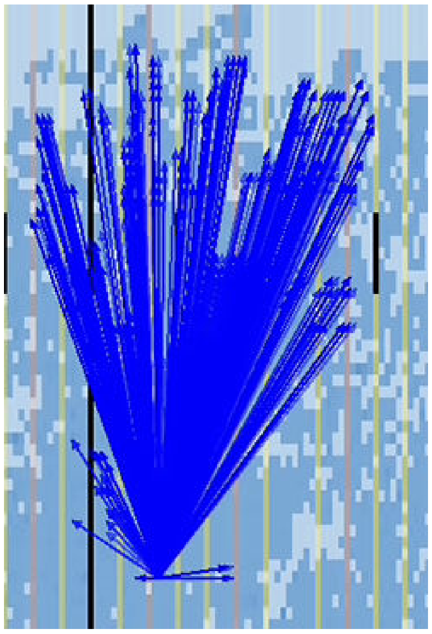
Non-Global High Fan-Out Signals

<<Filter>>

	Name	Fan-Out
1	uBP|regs_inst|registersLocal[0][96][0]	2965

3. To visualize the clock fan-out, right-click the signal name in the report, and then click **Locate Node ➤ Locate in Chip Planner**.

Figure 27. Non-Global High Fan-Out Signal in Chip Planner



4. To place those high fan-out signals on global resources, click **Assignments ➤ Assignment Editor**, and then assign the high fan-out signal to a global signal before re-starting compilation.

1.3.4. Enabling Physical Synthesis Optimization

Physical synthesis optimization improves circuit performance by performing combinational and sequential optimization and register duplication.

To enable physical synthesis options:

1. Click **Assignments ➤ Settings ➤ Compiler Settings**.
2. To enable retiming, combinational optimization, and register duplication, click **Advanced Settings (Fitter)**. Next, enable **Physical Synthesis**.
3. View physical synthesis results in the **Netlist Optimizations** report.

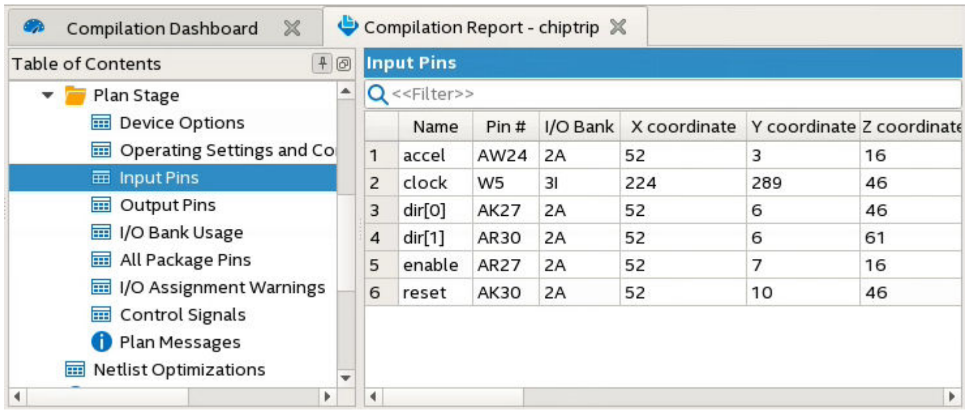
1.3.5. Viewing Fitter Reports

The Fitter generates detailed reports and messages for each stage of place and route. The Fitter Summary reports basic information about the Fitter run, such as date, software version, device family, timing model, and logic utilization.

1.3.5.1. Plan Stage Reports

The Plan stage reports describe the I/O, interface, and control signals discovered during the periphery planning stage of the Fitter.

Figure 28. Plan Stage Reports (Intel Arria 10 and Intel Cyclone 10 GX Designs)



Input Pins

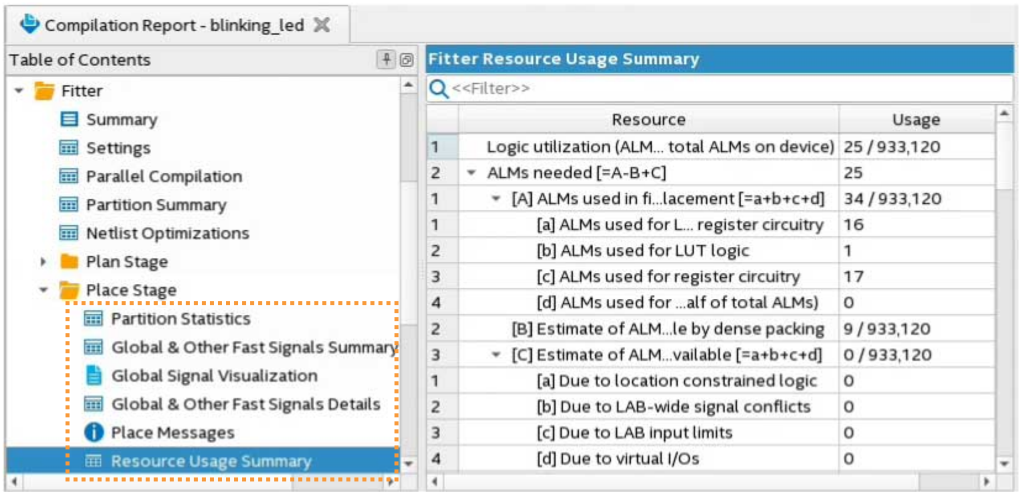
	Name	Pin #	I/O Bank	X coordinate	Y coordinate	Z coordinate
1	accel	AW24	2A	52	3	16
2	clock	W5	3I	224	289	46
3	dir[0]	AK27	2A	52	6	46
4	dir[1]	AR30	2A	52	6	61
5	enable	AR27	2A	52	7	16
6	reset	AK30	2A	52	10	46

For Intel Arria 10 and Intel Cyclone 10 GX designs, the Plan stage includes the **Global & Other Fast Signals Summary** report that allows you to verify which clocks the Compiler promotes to global clocks. Clock planning occurs after the Early Place stage for Intel Stratix 10 designs.

1.3.5.2. Place Stage Reports

The Place stage reports describe all device resources the Fitter allocates during logic placement, as well as use of Logic Lock regions and global and other fast signals.

Figure 29. Place Stage Reports



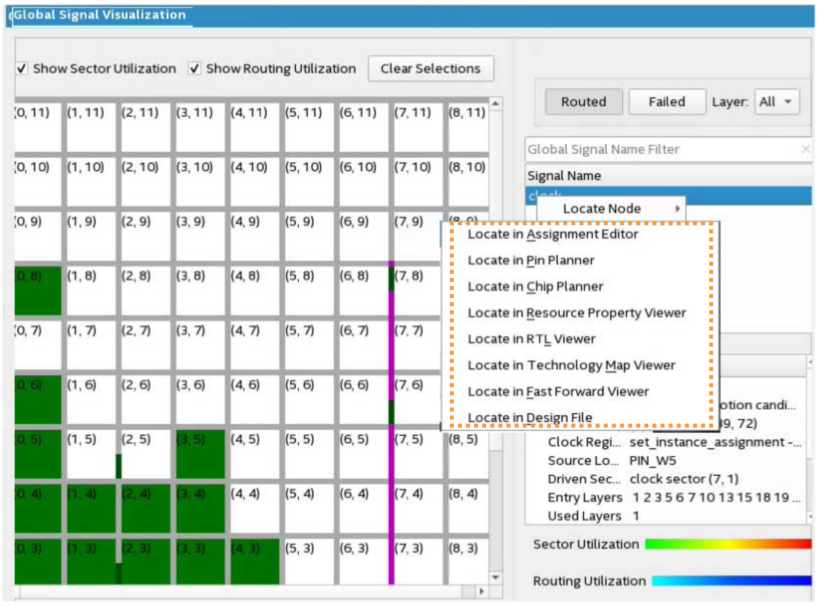
	Resource	Usage
1	Logic utilization (ALM... total ALMs on device)	25 / 933,120
2	ALMs needed [=A-B+C]	25
1	[A] ALMs used in fi...lacement [=a+b+c+d]	34 / 933,120
1	[a] ALMs used for L... register circuitry	16
2	[b] ALMs used for LUT logic	1
3	[c] ALMs used for register circuitry	17
4	[d] ALMs used for ...alf of total ALMs)	0
2	[B] Estimate of ALM...le by dense packing	9 / 933,120
3	[C] Estimate of ALM...vailable [=a+b+c+d]	0 / 933,120
1	[a] Due to location constrained logic	0
2	[b] Due to LAB-wide signal conflicts	0
3	[c] Due to LAB input limits	0
4	[d] Due to virtual I/Os	0

Global Signal Visualization Report

In addition, for Intel Stratix 10 designs, you can access the Global Signal Visualization report to view global signal routing and overall clock sector utilization in an interactive heat-map. Use this data to quickly review how the Fitter constrains clock regions to device sectors. View global clock tree implementation details and assess capacity to add more global signals to the design. In cases of clock tree synthesis errors, the report can also show targeted regions for failing signals, and competing signals that are contributing to routing congestion.

Filter the display to **Show Routing Utilization** and **Show Sector Utilization**. You can search for **Signal Names**, and then click the signal names to display its properties. Right-click the **Signal Names** to **Locate Node in Chip Planner** and various other tools.

Figure 30. Heat-Map in Global Signal Visualization Report

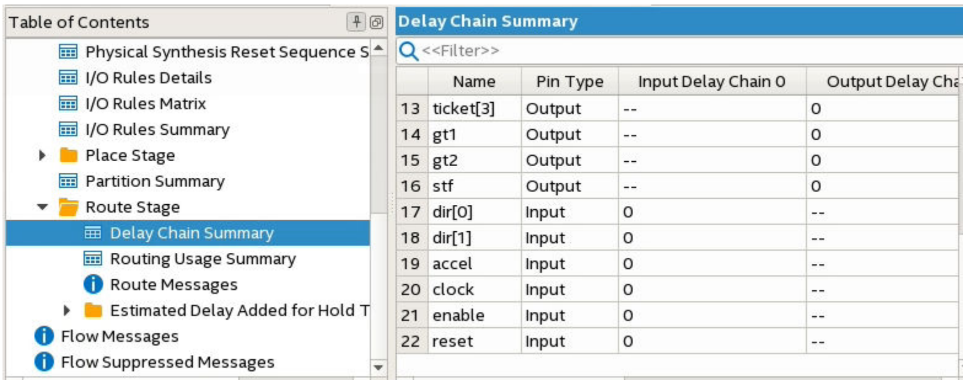


Clock Regi...	set_instance_assignment -...
Source Lo...	PIN_W5
Driven Sec...	clock sector (7, 1)
Entry Layers	1 2 3 5 6 7 10 13 15 18 19 ...
Used Layers	1

1.3.5.3. Route Stage Reports

The Route stage reports describe all device resources that the Fitter allocates during routing. Details include the type, number, and overall percentage of each resource type. The Route stage also reports delay chain summary information.

Figure 31. Route Stage Reports

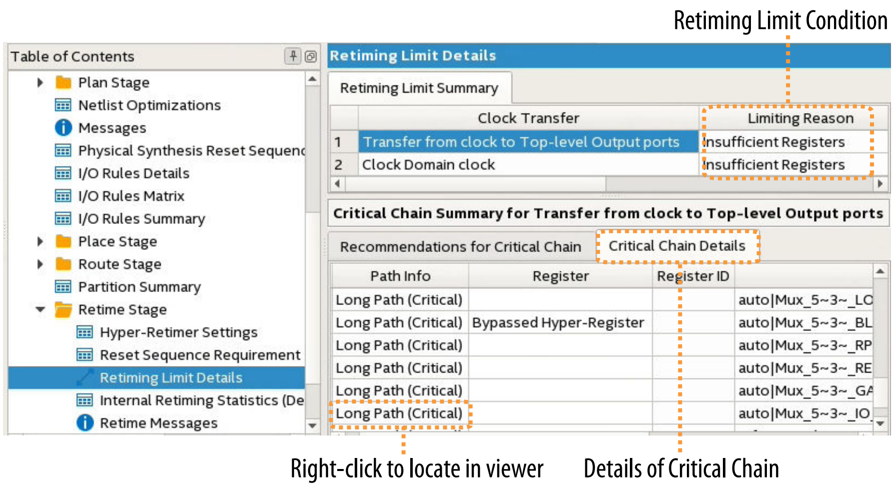


	Name	Pin Type	Input Delay Chain 0	Output Delay Cha
13	ticket[3]	Output	--	0
14	gt1	Output	--	0
15	gt2	Output	--	0
16	stf	Output	--	0
17	dir[0]	Input	0	--
18	dir[1]	Input	0	--
19	accel	Input	0	--
20	clock	Input	0	--
21	enable	Input	0	--
22	reset	Input	0	--

1.3.5.4. Retime Stage Reports

The Fitter generates detailed reports showing the results of the Retime stage, including the Retiming Limit Details report. This report lists the limiting reason, along with the critical chain and recommendations for the critical chain for each clock transfer.

Figure 32. Retiming Limit Details



	Clock Transfer	Limiting Reason
1	Transfer from clock to Top-level Output ports	Insufficient Registers
2	Clock Domain clock	Insufficient Registers

Critical Chain Summary for Transfer from clock to Top-level Output ports

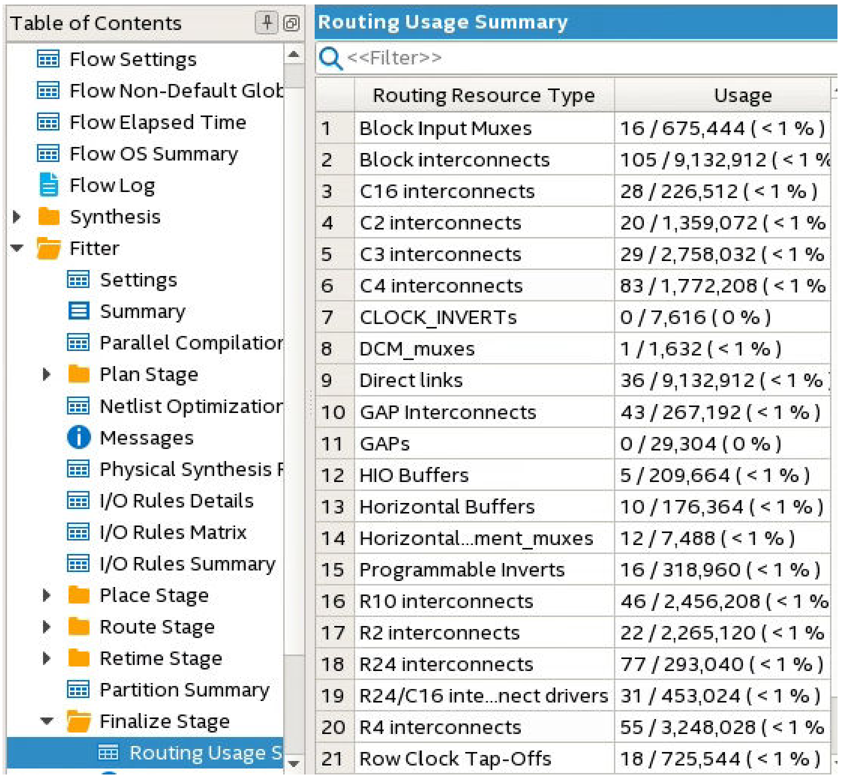
Path Info	Register	Register ID	
Long Path (Critical)			auto|Mux_5~3~_LO
Long Path (Critical)	Bypassed Hyper-Register		auto|Mux_5~3~_BL
Long Path (Critical)			auto|Mux_5~3~_RP
Long Path (Critical)			auto|Mux_5~3~_RE
Long Path (Critical)			auto|Mux_5~3~_GA
Long Path (Critical)			auto|Mux_5~3~_IO

1.3.5.5. Finalize Stage Reports

The Finalize stage reports describe final placement and routing operations, including:

- HSLP Summary. For Intel Arria 10 designs, the Compiler converts unnecessary tiles to High-Speed or Low-Power (HSLP) tiles.
- Post-route hold fix-up data. For Intel Stratix 10 designs, the Compiler reports hold violations for short paths following the Retime stage. The Fitter identifies and corrects the short paths with hold violations during the Fitter (Finalize) stage by adding routing wire along the paths.

Figure 33. Finalize Stage Reports (Intel Stratix 10 Design)



	Routing Resource Type	Usage
1	Block Input Muxes	16 / 675,444 (< 1 %)
2	Block interconnects	105 / 9,132,912 (< 1 %
3	C16 interconnects	28 / 226,512 (< 1 %)
4	C2 interconnects	20 / 1,359,072 (< 1 %
5	C3 interconnects	29 / 2,758,032 (< 1 %
6	C4 interconnects	83 / 1,772,208 (< 1 %
7	CLOCK_INVERTs	0 / 7,616 (0 %)
8	DCM_muxes	1 / 1,632 (< 1 %)
9	Direct links	36 / 9,132,912 (< 1 %)
10	GAP Interconnects	43 / 267,192 (< 1 %)
11	GAPs	0 / 29,304 (0 %)
12	HIO Buffers	5 / 209,664 (< 1 %)
13	Horizontal Buffers	10 / 176,364 (< 1 %)
14	Horizontal...ment_muxes	12 / 7,488 (< 1 %)
15	Programmable Inverts	16 / 318,960 (< 1 %)
16	R10 interconnects	46 / 2,456,208 (< 1 %
17	R2 interconnects	22 / 2,265,120 (< 1 %
18	R24 interconnects	77 / 293,040 (< 1 %)
19	R24/C16 inte...nect drivers	31 / 453,024 (< 1 %)
20	R4 interconnects	55 / 3,248,028 (< 1 %
21	Row Clock Tap-Offs	18 / 725,544 (< 1 %)

Related Information

Step 2: Review Retiming Results on page 40
For information on Retiming and Fast Forward compilation reports

1.4. Running Full Compilation

Use these steps to run a full compilation of an Intel Quartus Prime project. A full compilation includes IP Generation, Analysis & Synthesis, Fitter, Timing Analyzer, and any optional Compiler modules you enable.

1. Before running a full compilation, specify any of the following project settings:
   - To specify the target FPGA device or development kit, click **Assignments ➤ Device**.
   - To specify device and pin options for the target FPGA device, click **Assignments ➤ Device ➤ Device and Pin Options**.
   - To specify options that affect compilation processing time and netlist preservation, click **Assignments ➤ Settings ➤ Compilation Process Settings**.

- To specify the Compiler's high-level optimization strategy, click **Assignments ➤ Settings ➤ Compiler Settings**. Specify a **Balanced** strategy, or optimize for **Performance**, **Area**, **Routability**, **Power**, or **Compile Time**. The Compiler targets the optimization goal you specify. Optimization Modes on page 70 describes these options in detail.
- To specify synthesis algorithm and other **Advanced Settings** for synthesis and fitting, click **Assignments ➤ Settings ➤ Compiler Settings**. Turn on **Enable Intermediate Fitter Snapshots** to preserve the planned, placed, routed, and retimed snapshots by default during full compilation.
- To specify required timing conditions for proper operation of your design, click **Tools ➤ Timing Analyzer**.

2. To run full compilation, click **Processing ➤ Start Compilation**.

   Note:
   - To save processing time, the Compiler only preserves the planned, placed, routed, and retimed snapshots by default during full compilation if you turn on **Enable Intermediate Fitter Snapshots** (**Assignments ➤ Settings ➤ Compiler Settings**).
   - Early Place does not run during full compilation by default. To enable Early Place during full compilation, click **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Fitter)** to modify the **Run Early Place during compilation** option.

Related Information

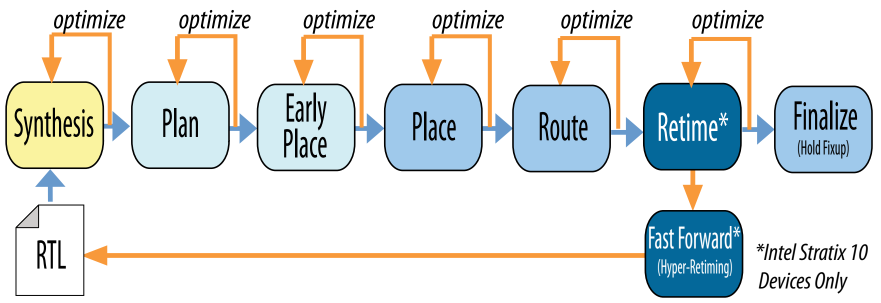
- Intel Quartus Prime Pro Edition User Guide: Design Constraints
- Intel Quartus Prime Pro Edition User Guide: Timing Analyzer

1.5. Running the Hyper-Aware Design Flow

The Intel Quartus Prime Pro Edition Compiler helps you to take full advantage of the Intel Stratix 10 Intel Hyperflex™ architecture. Use the Hyper-Aware design flow to shorten design cycles and optimize performance.

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

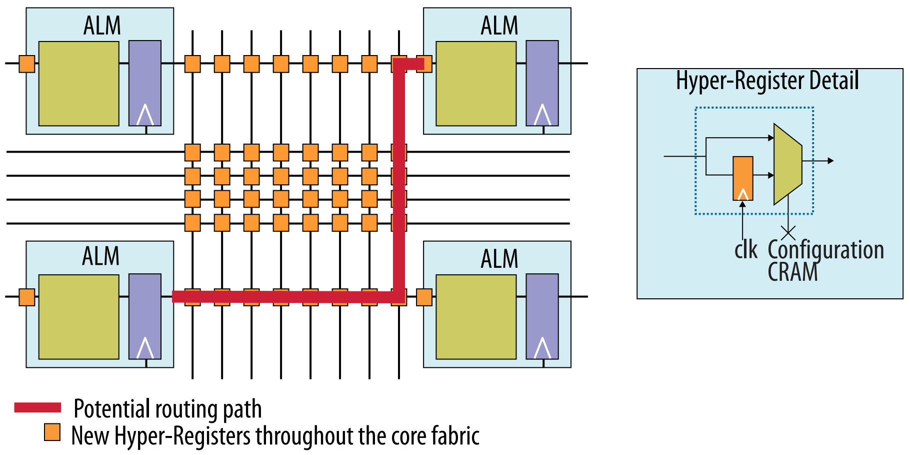
Figure 34. Hyper-Aware Design Flow



Hyper-Retiming

A key innovation of the Intel Stratix 10 architecture is the addition of multiple Hyper-Registers in every routing segment and block input. Maximizing the use of Hyper-Registers improves design performance. The prevalence of Hyper-Registers improves balance of time delays between registers and mitigates critical path delays. Hyper-Retiming moves registers out of ALMs and retimes them into Hyper-Registers, wherever advantageous. Hyper-Retiming runs automatically during fitting, requires minimal effort, and can result in significant performance improvement.

Figure 35. Hyper-Register Architecture



Fast Forward Compilation

If you require optimization beyond Hyper-Retiming, run Fast Forward compilation to generate timing closure recommendations that break key performance bottlenecks. Fast Forward compilation shows precisely where to make the most impact with RTL changes, and reports the performance benefits you can expect from each change. The Fitter does not automatically retime registers across RAM and DSP blocks. However, Fast Forward analysis shows the potential performance benefit from this optimization.

Fast-Forward compilation identifies the best location to add pipeline stages (Hyper-Pipelining), and the expected performance benefit in each case. After you modify the RTL to place pipeline stages at the boundaries of each clock domain, the Hyper-Retimer automatically places the registers within the clock domain at the optimal locations to maximize performance. Implement the recommendations in RTL to achieve similar results. After implementing any changes, re-run the Hyper-Retimer until the results meet performance and timing requirements. Fast Forward compilation does not run automatically as part of a full compilation. Enable or run **Fast Forward compilation** in the Compilation Dashboard.

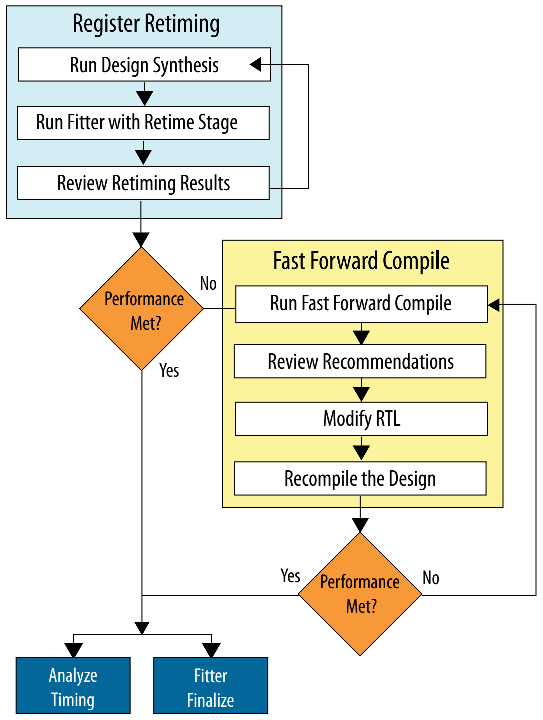
Table 8. HyperFlex Optimization Steps

Optimization Step	Technique	Description
Step 1	Hyper-Retiming	Retimer moves existing registers into Hyper-Registers.
Step 2	Fast Forward Compile	Compiler generates design-specific timing closure recommendations and predicts performance improvement.
Step 3	Hyper-Pipelining	Use Fast Forward compilation to identify where to add new registers and pipeline stages in RTL.
Step 4	Hyper-Optimization	Design optimization beyond Hyper-Retiming and Hyper-Pipelining, such as restructuring loops, removing control logic limits, and reducing the delay along long paths.

The Hyper-Aware design flow includes the following high-level steps this chapter covers in detail:

1. Run the Retime stage during the Fitter to automatically retime ALM registers into Hyper-Registers.
2. Review Retiming Results in the Compilation Report.
3. If you require further performance optimization, run Fast Forward compilation.
4. Review Fast Forward timing closure recommendations.
5. Implement appropriate Fast Forward recommendations in your RTL.
6. Recompile the design through the Retime stage.

Figure 36. Hyper-Aware Design Flow



1.5.1. Step 1: Run Register Retiming

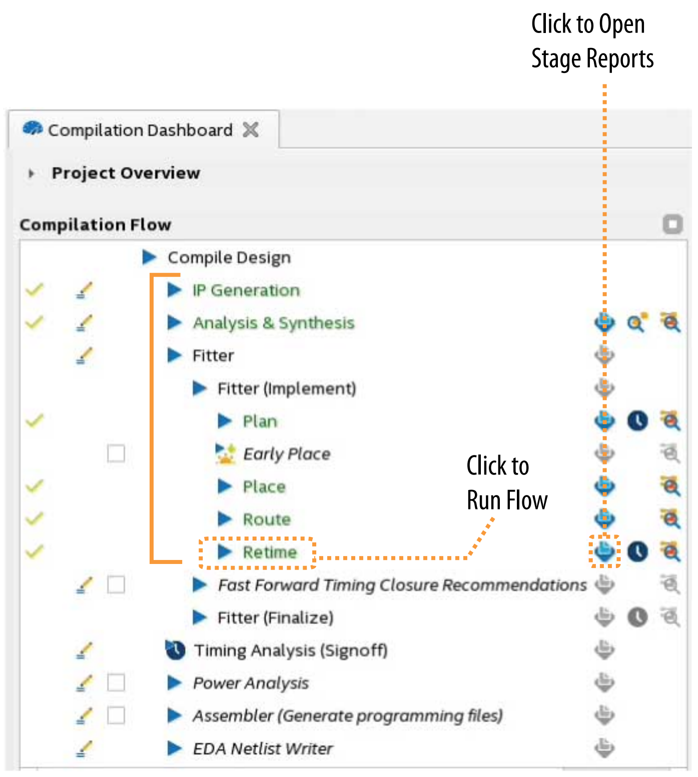
Register retiming improves design performance by moving registers out of ALMs and retimes them into Hyper-Registers in the Intel Stratix 10 device interconnect.

The Fitter runs the **Retime** stage automatically following place and route when you target an Intel Stratix 10 device. Alternatively, start or stop the individual **Retime** stage in the Compilation Dashboard. After running register retiming, view the Fitter reports to optimize remaining critical paths.

To run register retiming:

1. Create or open an Intel Quartus Prime project that is ready for design synthesis and fitting.
2. To run register retiming, click **Retime** on the Compilation Dashboard. The Compiler runs prerequisite stages automatically, as needed. The Compiler generates detailed reports and timing analysis data for each stage. Click the **Report** or **Timing Analyzer** icons to review results of each stage. Rerun any stage to apply any setting or design changes.
3. If register retiming achieves all performance goals for your design, proceed to Fitter (Finalize) and timing analysis stages of compilation. If your design requires further optimization, run **Fast Forward Timing Closure Recommendations**.

Figure 37. Retiming Stage in Compilation Dashboard



1.5.2. Step 2: Review Retiming Results

The Fitter generates detailed reports showing the results of the Retime stage. Follow these steps to review the results and make additional performance improvements with register retiming.

1. To open the **Retiming Limit Details** report, click the **Report** icon for the Retime stage in the Compilation Dashboard. The **Retiming Limit Details** lists the number of registers moved, their paths, and the limiting reason preventing further retiming.
2. To further optimize, resolve any **Limiting Reason** in your design, and then rerun the **Retime** stage, as necessary.
3. If register retiming achieves all performance goals for your design, proceed to Fitter (Finalize) and Timing Analysis stages of compilation.
4. If your design requires further optimization, run **Fast Forward Timing Closure Recommendations**.

Table 9. Retiming Limit Details Report Data

Report Data	Description
Clock Transfer	Lists each clock domain in your design. Click the domain to display data about each entry.
Limiting Reason	Specifies any design condition that prevent further register retiming improvement, such as any of the following conditions: • **Insufficient Registers**—indicates insufficient quantity of registers at either end of the chain for retiming. Adding more registers can improve performance. • **Short Path/Long Path**—indicates that the critical chain has dependent paths with conflicting characteristics. For example, one path improves performance with more registers, and another path has no place for additional registers. • **Path Limit**—indicates that there are no further Hyper-Register locations available on the critical path, or the design reached a performance limit of the current place and route. • **Loops**—indicates a feedback path in a circuit. When the critical chain includes a feedback loop, retiming cannot change the number of registers in the loop without changing functionality. The Compiler can retime around the loop without changing functionality. However, the Compiler cannot place additional registers in the loop.
Critical Chain Details	Lists register timing path associated with the retiming limitations. Right-click any path to **Locate Critical Chain in Technology Map Viewer**.

Figure 38. Retiming Limit Details

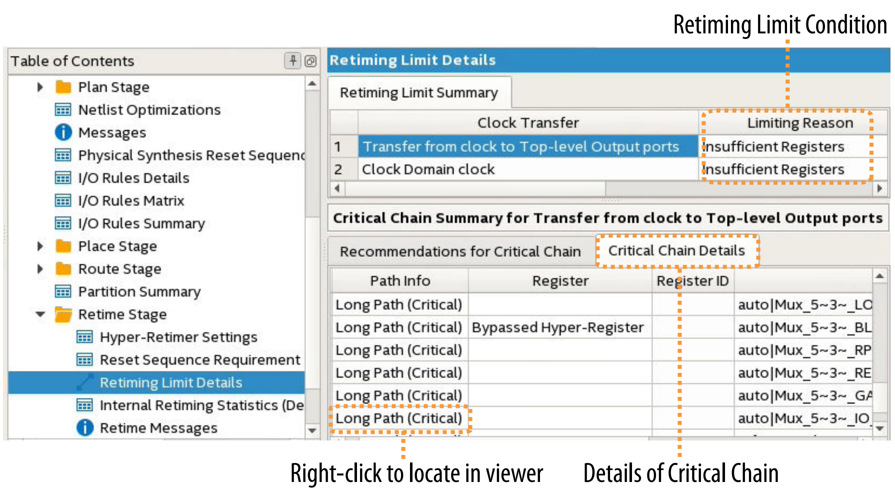


Table of Contents
- Plan Stage
- Netlist Optimizations
- Messages
- Physical Synthesis Reset Sequenc
- I/O Rules Details
- I/O Rules Matrix
- I/O Rules Summary
- Place Stage
- Route Stage
- Partition Summary
- Retime Stage
  - Hyper-Retimer Settings
  - Reset Sequence Requirement
  - Retiming Limit Details
  - Internal Retiming Statistics (De
  - Retime Messages

Retiming Limit Details

Retiming Limit Summary

	Clock Transfer	Limiting Reason
1	Transfer from clock to Top-level Output ports	Insufficient Registers
2	Clock Domain clock	Insufficient Registers

Critical Chain Summary for Transfer from clock to Top-level Output ports

Recommendations for Critical Chain | Critical Chain Details

Path Info	Register	Register ID	
Long Path (Critical)			auto|Mux_5~3~_LO
Long Path (Critical)	Bypassed Hyper-Register		auto|Mux_5~3~_BL
Long Path (Critical)			auto|Mux_5~3~_RP
Long Path (Critical)			auto|Mux_5~3~_RE
Long Path (Critical)			auto|Mux_5~3~_GA
Long Path (Critical)			auto|Mux_5~3~_IO

Note: The Compiler reports any hold violations for short paths following the Retime stage. The Fitter identifies and corrects the short paths with hold violations during the Fitter (Finalize) stage by adding routing wire along the paths.

1.5.2.1. Locate Critical Chains

The **Retiming Limit Details** reports the design paths that limit further register retiming. Right-click any path to locate to the path in the Technology Map Viewer - Post-fitting view. This viewer displays a schematic representation of the complete design after place, route, and register retiming. To view the retimed netlist in the Technology Map Viewer, follow these steps:

1. To open the **Retiming Limit Details** report, click the **Report** icon next to the **Retime** stage in the Compilation Dashboard.
2. Right-click any path in the **Retiming Limit Details** report and click **Locate Critical Chain in Technology Map Viewer**. The netlist displays as a schematic in the Technology Map Viewer.

Figure 39. Technology Map Viewer

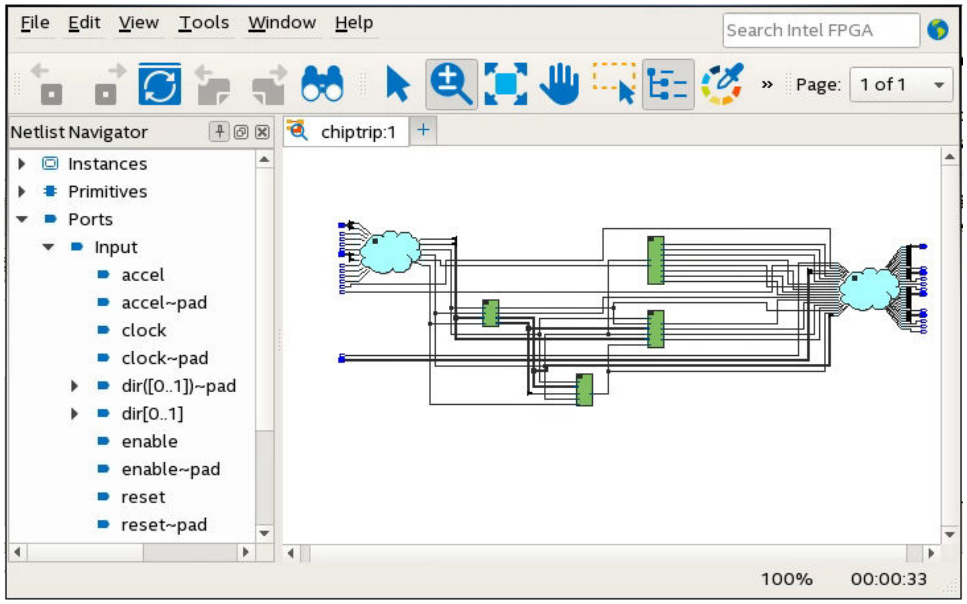
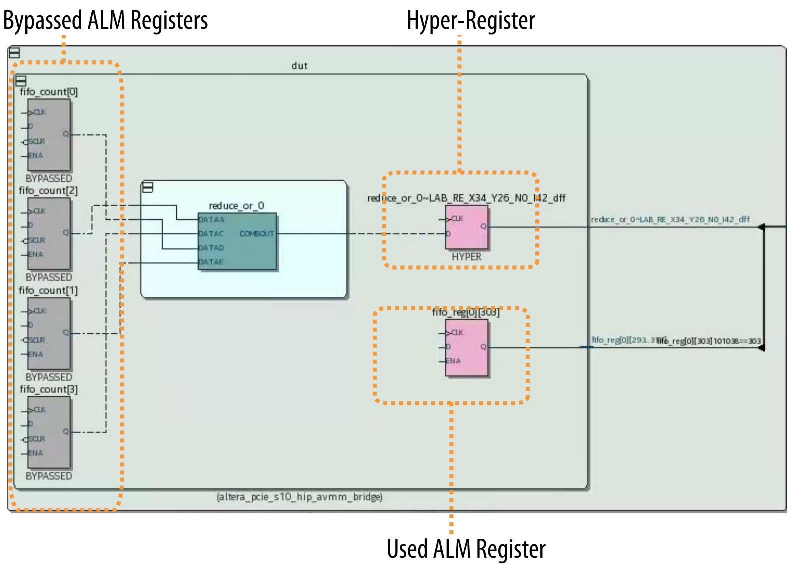


Figure 40. Post-Fit Viewer After Retiming



1.5.3. Step 3: Run Fast Forward Compile and Hyper-Retiming

When you run Fast Forward compilation, the Compiler predictively removes signals from registers to allow mobility within the netlist for subsequent retiming. Fast Forward compilation generates design-specific timing closure recommendations, and predicts maximum performance with removal of all timing restrictions. After you complete Fast Forward explorations, determine which recommendations you can implement to provide the most benefit. Implement appropriate recommendations in your RTL, and recompile the design to realize the performance levels that Fast Forward reports.

To generate Fast Forward timing closure recommendations, follow these steps:

1. On the Compilation Dashboard, click **Fast Forward Timing Closure Recommendations**. The Compiler runs prerequisite synthesis or Fitter stages automatically, as needed, and generates timing closure recommendations in the Compilation Report.
2. View timing closure recommendations in the Compilation Report to evaluate design performance and implement key RTL performance improvements.
3. Optionally, specify any of the following any of the following options if you want to automate or refine Fast Forward analysis:
   - If you want to run Fast Forward compilation during each full compilation, click **Assignments ➤ Settings ➤ Compiler Settings ➤ HyperFlex** and enable **Run Fast Forward Timing Closure Recommendations during compilation.**
   - If you want to modify how Fast Forward compilation interprets specific I/O and block types, click **Assignments ➤ Settings ➤ Compiler Settings ➤ HyperFlex ➤ Advanced Settings**.

Figure 41. Running Fast Forward Compilation

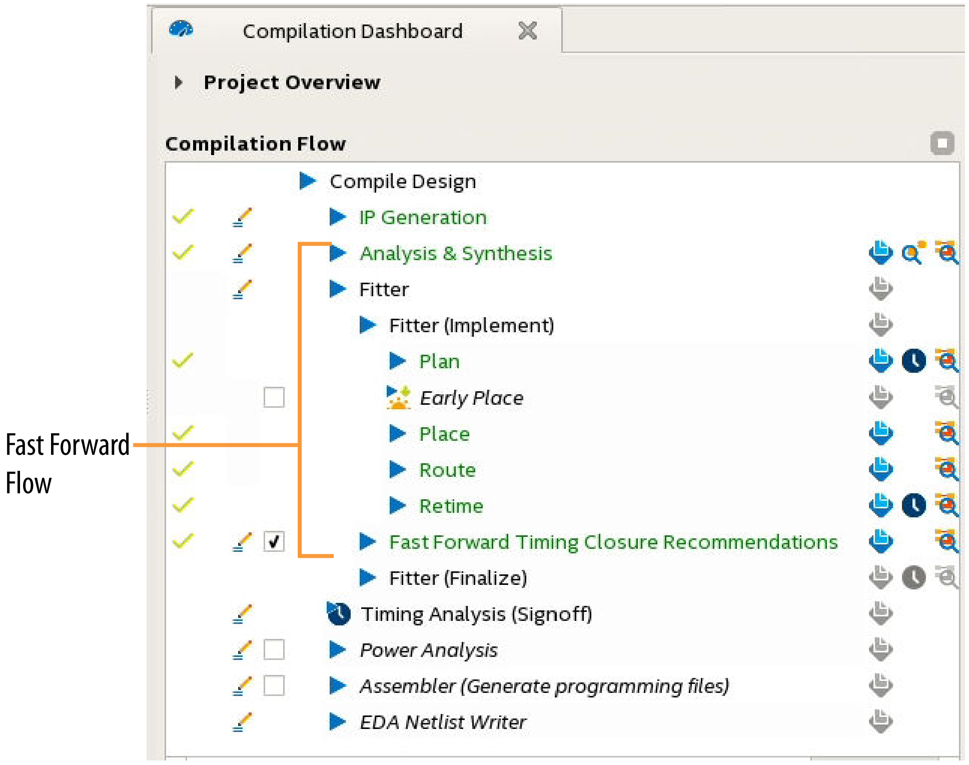
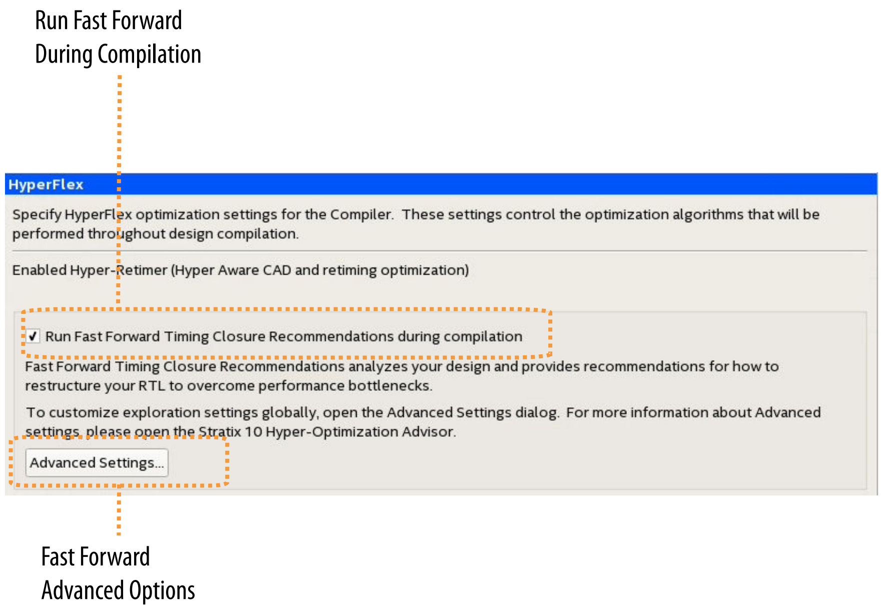


Figure 42. HyperFlex Settings


Run Fast Forward During Compilation
HyperFlex
Advanced Settings...
Fast Forward Advanced Options

Specify HyperFlex optimization settings for the Compiler. These settings control the optimization algorithms that will be performed throughout design compilation.

Enabled Hyper-Retimer (Hyper Aware CAD and retiming optimization)

[✓] Run Fast Forward Timing Closure Recommendations during compilation

Fast Forward Timing Closure Recommendations analyzes your design and provides recommendations for how to restructure your RTL to overcome performance bottlenecks.

To customize exploration settings globally, open the Advanced Settings dialog. For more information about Advanced settings, please open the Stratix 10 Hyper-Optimization Advisor.

1.5.3.1. HyperFlex Settings

The **HyperFlex** settings page controls whether Fast Forward Compilation analyzes and reports results for specific logical structures in the Intel Hyperflex architecture of the Intel Stratix 10 FPGA. You access this page by clicking **Assignments ➤ Settings ➤ HyperFlex**. Turn on **Run Fast Forward Timing Closure Recommendations during compilation** to enable Fast Forward analysis during the compilation flow by default. To access the following additional settings, click **Advanced Settings**.

Table 10. Advanced HyperFlex Settings

Option	Description
Fast Forward Compile Asynchronous Clears	Specifies how Fast Forward analysis accounts for registers with asynchronous clear signals. The options are: • **Auto**—the Compiler identifies asynchronous clears as asynchronous until they limit timing performance during Fast Forward Compilation, at which point the Compiler identifies the asynchronous clears as removed. • **Preserve**—the Compiler never assumes removal or conversion of asynchronous clears for Fast Forward analysis.
Fast Forward Compile Cut All Clock Transfers	Cuts all clock transfers in Fast Forward Compilation analysis.
Fast Forward Compile Fully Registered DSP Blocks	Specifies how Fast Forward analysis accounts for DSP blocks that limit performance. Enable this option to generate results as if all DSP blocks are fully registered.
Fast Forward Compile Fully Registered RAM Blocks	Specifies how Fast Forward analysis accounts for RAM blocks that limit performance. Enable this option to analyze the blocks as fully registered.
Fast Forward Compile Maximum Additional Pipeline Stages	Specifies the maximum number of pipeline stages that Fast Forward compilation explores.
Fast Forward Compile User Preserve Directives	Specifies how Fast Forward compilation accounts for restrictions from user-preserve directives.

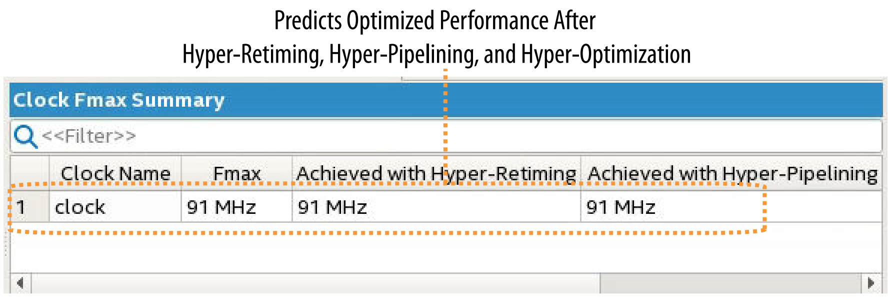
1.5.4. Step 4: Review Hyper-Retiming Results

After running Fast Forward Compilation, review the reports in the Fast Forward Timing Closure Recommendations folder of the Compilation Report to determine which recommendations are appropriate and practical for your design functionality and performance goals.

1.5.4.1. Clock Fmax Summary Report

The Clock Fmax Summary in the **Fast Forward Timing Closure Recommendations** report folder reports the current f_{max} and potential performance achievable for each clock domain after Hyper-Retiming, Hyper-Pipelining, and Hyper-Optimization steps. Review the Clock Fmax Summary data to determine whether each potential performance improvement warrants further investigation and potential optimization of design RTL.

Figure 43. Current and Potential Performance in Clock Fmax Summary


Predicts Optimized Performance After Hyper-Retiming, Hyper-Pipelining, and Hyper-Optimization
Clock Fmax Summary
<<Filter>>

	Clock Name	Fmax	Achieved with Hyper-Retiming	Achieved with Hyper-Pipelining
1	clock	91 MHz	91 MHz	91 MHz

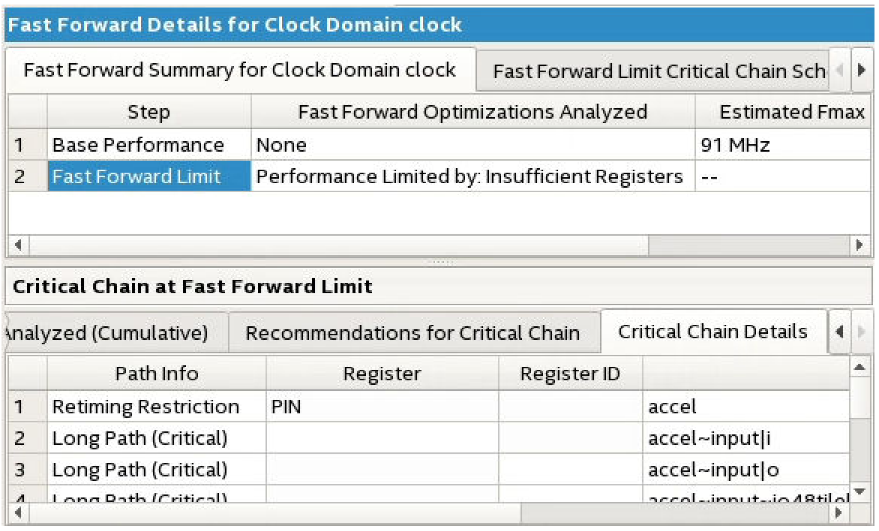
1.5.4.2. Fast Forward Details Report

The Fast Forward Details report recommends the design modifications necessary to achieve Fast Forward compilation performance levels. Some recommendations may be functionally impossible or impractical for your design. Consider which recommendations you can implement in RTL to achieve similar performance improvement. Click any optimization **Step** to view the implementation details and performance calculations for that step.

Table 11. Fast Forward Details Report Data

Report Field	Description
Step	Displays the pre-optimized Base Performance f_{MAX}, the recommended Fast Forward optimization steps, and the Fast Forward Limit critical path that prevents further optimization.
Fast Forward Optimizations Analyzed	Summarizes the optimizations necessary to implement each optimization step.
Estimated Fmax	Specifies the potential f_{MAX} performance if you implement all Fast Forward optimization steps.
Optimizations Analyzed For Fast Forward Step	Lists design recommendations hierarchically for the selected **Step**. Click the text to expand the report and view the clock domain, the affected module, and the bus and bits that require modification.
Optimizations Analyzed (Cumulative)	Accumulated list of all design changes necessary to reach the selected **Step**.
Critical Chain at Fast Forward Limit	Displays information about any path that continues to limit Hyper-Retiming even after application of all Fast Forward steps. The critical chain is any path that limits further Hyper-Retiming. Click the **Fast Forward Limit** step to display this field.
Recommendations for Critical Chain	Lists register timing path associated with the retiming limitations. Right-click any path to **Locate Critical Chain in Fast Forward Viewer**.

Figure 44. Fast-Forward Details Report



Fast Forward Details for Clock Domain clock

Fast Forward Summary for Clock Domain clock | Fast Forward Limit Critical Chain Sch

	Step	Fast Forward Optimizations Analyzed	Estimated Fmax
1	Base Performance	None	91 MHz
2	Fast Forward Limit	Performance Limited by: Insufficient Registers	--

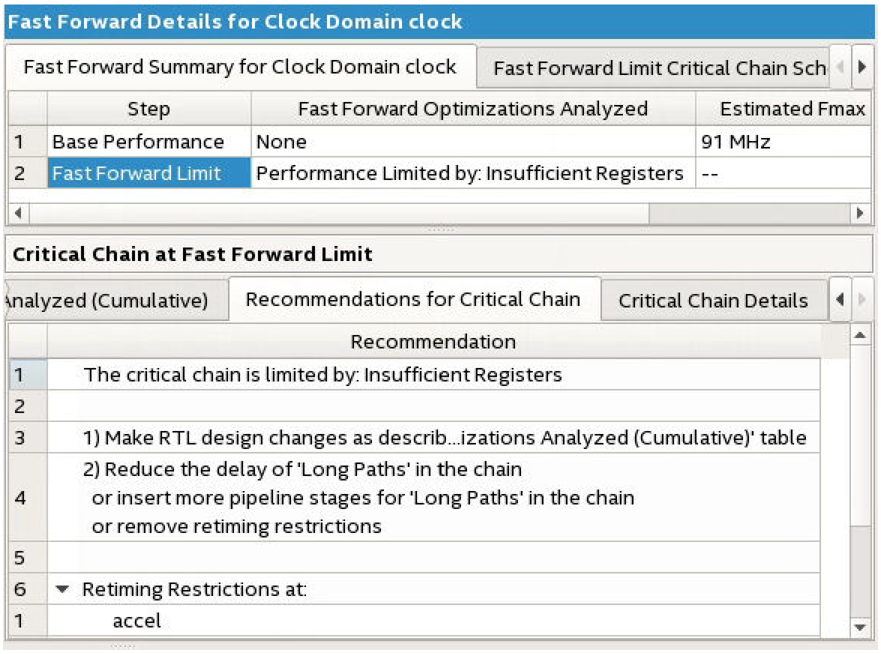
Critical Chain at Fast Forward Limit

nalyzed (Cumulative) | Recommendations for Critical Chain | Critical Chain Details

	Path Info	Register	Register ID	
1	Retiming Restriction	PIN		accel
2	Long Path (Critical)			accel~input|i
3	Long Path (Critical)			accel~input|o
4	Long Path (Critical)			accel~input~io48tilel

Right-click any path to locate to the critical chain in the Fast Forward Viewer. The Fast Forward Viewer displays a predictive representation of the complete design, after implementation of all Fast Forward recommendations.

Figure 45. Recommendations for Critical Chain



Fast Forward Details for Clock Domain clock

Fast Forward Summary for Clock Domain clock | Fast Forward Limit Critical Chain Sch

	Step	Fast Forward Optimizations Analyzed	Estimated Fmax
1	Base Performance	None	91 MHz
2	Fast Forward Limit	Performance Limited by: Insufficient Registers	--

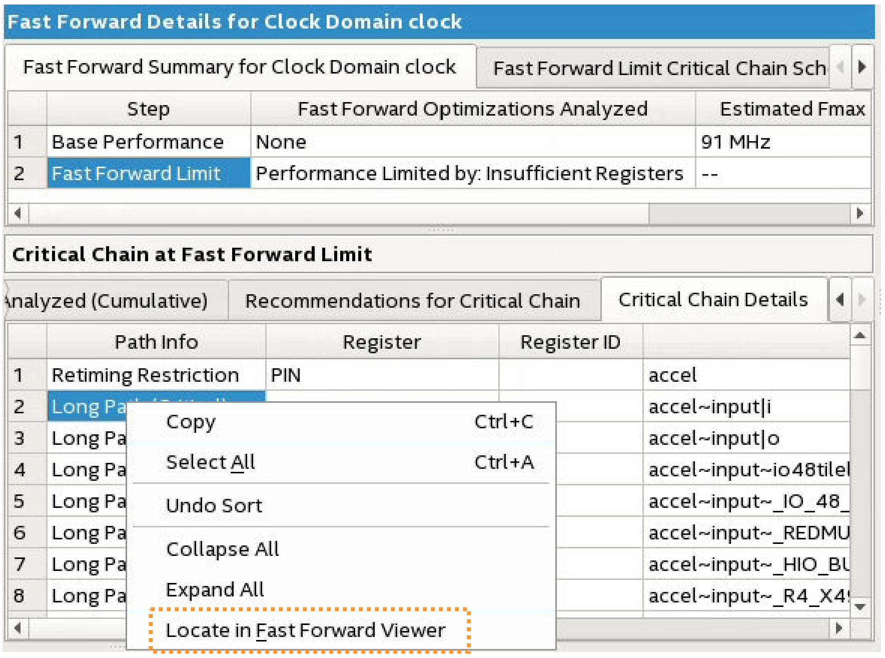
Critical Chain at Fast Forward Limit

nalyzed (Cumulative) | Recommendations for Critical Chain | Critical Chain Details

	Recommendation
1	The critical chain is limited by: Insufficient Registers
2	
3	1) Make RTL design changes as describ...izations Analyzed (Cumulative)' table
4	2) Reduce the delay of 'Long Paths' in the chain or insert more pipeline stages for 'Long Paths' in the chain or remove retiming restrictions
5	
6	▼ Retiming Restrictions at:
1	accel

Send Feedback

Figure 46. Locate Critical Chain in Fast Forward Viewer



Fast Forward Details for Clock Domain clock

Fast Forward Summary for Clock Domain clock | Fast Forward Limit Critical Chain Sch

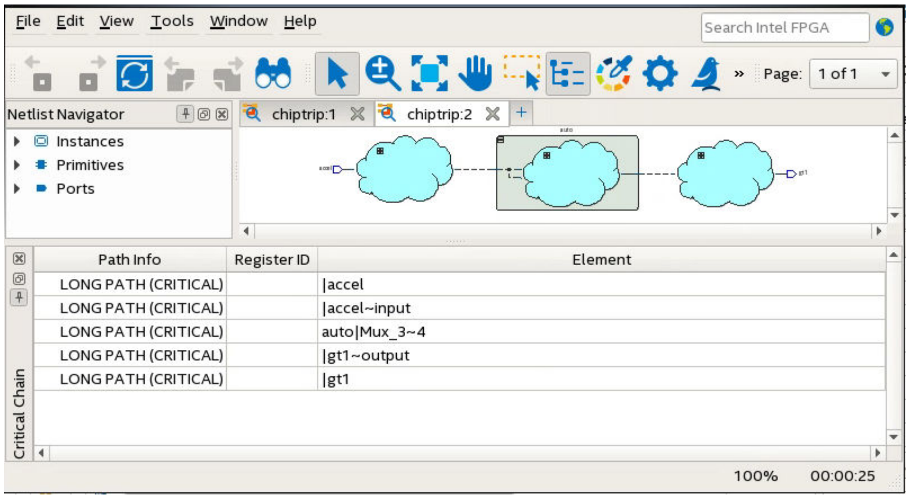
	Step	Fast Forward Optimizations Analyzed	Estimated Fmax
1	Base Performance	None	91 MHz
2	Fast Forward Limit	Performance Limited by: Insufficient Registers	--

Critical Chain at Fast Forward Limit

nalyzed (Cumulative) | Recommendations for Critical Chain | Critical Chain Details

	Path Info	Register	Register ID	
1	Retiming Restriction	PIN		accel
2	Long Pa			accel~input|i
3	Long Pa			accel~input|o
4	Long Pa			accel~input~io48tilel
5	Long Pa			accel~input~_IO_48_
6	Long Pa			accel~input~_REDMU
7	Long Pa			accel~input~_HIO_BU
8	Long Pa			accel~input~_R4_X4

Figure 47. Fast Forward Viewer Shows Predictive Results



Path Info	Register ID	Element
LONG PATH (CRITICAL)		|accel
LONG PATH (CRITICAL)		|accel~input
LONG PATH (CRITICAL)		auto|Mux_3~4
LONG PATH (CRITICAL)		|gt1~output
LONG PATH (CRITICAL)		|gt1

1.5.5. Step 5: Implement Fast Forward Recommendations

Implement the Fast Forward timing closure recommendations in your design RTL and rerun the **Retime** stage to realize the predictive performance gains. The amount and type of changes that you implement depends on your performance goals. For example, if you can achieve the target f_{MAX} with simple asynchronous clear removal or conversion, you can stop design optimization after making those changes. However, if you require additional performance, implement more Fast Forward recommendations, such as any of the following techniques:

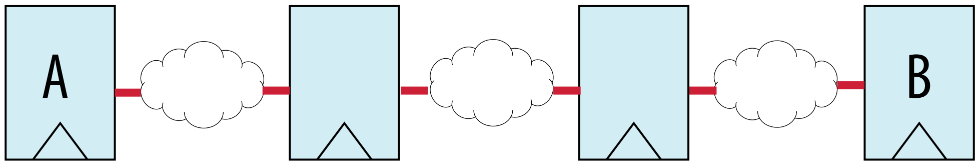
- Remove limitations of control logic, such as long feedback loops and state machines.
- Restructure logic to use functionally equivalent feed-forward or pre-compute paths, rather than long combinatorial feedback path.
- Reduce the delay of 'Long Paths' in the chain. Use standard timing closure techniques to reduce delay. Excessive combinational logic, sub-optimal placement, and routing congestion cause delay on paths.
- Insert more pipeline stages in 'Long Paths' in the chain. Long paths have the most delay between registers in the critical chain.
- Increase the delay (or add pipeline stages to 'Short Paths' in the chain).
- Explore performance and implement the RTL changes to your code until you reach the desired performance target.

1.5.5.1. Retiming Restrictions and Workarounds

The Compiler identifies the register chains in your design that limit further optimization through Hyper-Retiming. The Compiler refers to these related register-to-register paths as a critical chain. The f_{MAX} of the critical chain and its associated clock domain is limited by the average delay of a register-to-register path, and quantization delays of indivisible circuit elements like routing wires. There are a variety of situations that cause retiming restrictions. Retiming restrictions exist because of hardware characteristics, software behavior, or are inherent to the design. The **Retiming Limit Details** report the limiting reasons preventing further retiming, and the registers and combinational nodes that comprise the chain. The Fast Forward recommendations list the steps you can take to remove critical chains and enable additional register retiming.

In Figure 48 on page 48, the red line represents the same critical chain. Timing restrictions prevent register A from retiming forward. Timing restrictions also prevent register B from retiming backwards. A loop occurs when register A and register B are the same register.

Figure 48. Sample Critical Chain



Fast Forward recommendations for the critical chain include:

- Reduce the delay of 'Long Paths' in the chain. Use standard timing closure techniques to reduce delay. Combinational logic, sub-optimal placement, and routing congestion, are among the reasons for path delay.
- Insert more pipeline stages in 'Long Paths' in the chain. Long paths are the parts of the critical chain that have the most delay between registers.
- Increase the delay (or add pipeline stages to 'Short Paths' in the chain).

Particular registers in critical chains can limit performance for many other reasons. The Compiler classifies the following types of reasons that limit further optimization by retiming:

- Insufficient Registers
- Loop
- Short path/long path
- Path limit

After understanding why a particular critical chain limits your design's performance, you can then make RTL changes to eliminate that bottleneck and increase performance.

Table 12. Hyper-Register Support for Various Design Conditions

Design Condition	Hyper-Register Support
Initial conditions that cannot be preserved	Hyper-Registers do have initial condition support. However, you cannot perform some retiming operations while preserving the initial condition stage of all registers (that is, the merging and duplicating of Hyper-Registers). If this condition occurs in the design, the Fitter does not retime those registers. This retiming limit ensures that the register retiming does not affect design functionality.
Register has an asynchronous clear	Hyper-Registers support only data and clock inputs. Hyper-Registers do not have control signals such as asynchronous clears, presets, or enables. The Fitter cannot retime any register that has an asynchronous clear. Use asynchronous clears only when necessary, such as state machines or control logic. Often, you can avoid or remove asynchronous clears from large parts of a datapath.
Register drives an asynchronous signal	This design condition is inherent to any design that uses asynchronous resets. Focus on reducing the number of registers that are reset with an asynchronous clear.
Register has don't touch or preserve attributes	The Compiler does not retime registers with these attributes. If you use the `preserve` attribute to manage register duplication for high fan-out signals, try removing the `preserve` attribute. The Compiler may be able to retime the high fan-out register along each of the routing paths to its destinations. Alternatively, use the `dont_merge` attribute. The Compiler retimes registers in ALMs, DDIOs, single port RAMs, and DSP blocks.
Register is a clock source	This design condition is uncommon, especially for performance-critical parts of a design. If this retiming restriction prevents you from achieving the required performance, consider whether a PLL can generate the clock, rather than a register.
Register is a partition boundary	This condition is inherent to any design that uses design partitions. If this retiming restriction prevents you from achieving the required performance, add additional registers inside the partition boundary for Hyper-Retiming.
Register is a block type modified by an ECO operation	This restriction is uncommon. Avoid the restriction by making the functional change in the design source and recompiling, rather than performing an ECO.
Register location is an unknown block	This restriction is uncommon. You can often work around this condition by adding extra registers adjacent to the specified block type.
Register is described in the RTL as a latch	Hyper-Registers cannot implement latches. The Compiler infers latches because of RTL coding issues, such as incomplete assignments. If you do not intend to implement a latch, change the RTL.
Register location is at an I/O boundary	All designs contain I/O, but you can add additional pipeline stages next to the I/O boundary for Hyper-Retiming.

continued...

Design Condition	Hyper-Register Support
Combinational node is fed by a special source	This condition is uncommon, especially for performance-critical parts of a design.
Register is driven by a locally routed clock	Only the dedicated clock network clocks Hyper-Registers. Using the routing fabric to distribute clock signals is uncommon, especially for performance-critical parts of a design. Consider implementing a small clock region instead.
Register is a timing exception end-point	The Compiler does not retime registers that are sources or destinations of `.sdc` constraints.
Register with inverted input or output	This condition is uncommon.
Register is part of a synchronizer chain	The Fitter optimizes synchronizer chains to increase the mean time between failure (MTBF), and the Compiler does not retime registers that are detected or marked as part of a synchronizer chain. Add more pipeline stages at the clock domain boundary adjacent to the synchronizer chain to provide flexibility for the retiming.
Register with multiple period requirements for paths that start or end at the register (cross-clock boundary)	This situation occurs at any cross-clock boundary, where a register latches data on a clock at one frequency, and fans out to registers running at another frequency. The Compiler does not retime registers at cross-clock boundaries. Consider adding additional pipeline stages at one side of the clock domain boundary, or the other, to provide flexibility for retiming.

1.6. Exporting Compilation Results

When you run compilation, the Compiler preserves a database of results in a Quartus Database File (`.qdb`). The `.qdb` contains the data to reproduce similar results in another project, or in a later software version. You can export your project's compilation results database for import to another project or migration to a later Intel Quartus Prime software version.

You can export the `.qdb` for your entire project or for a design partition that you define in your project. When migrating the database for an entire project, you can export the compilation database in a *version-compatible* format to ensure compatibility for import to a later software version. Although you cannot directly read the contents of the `.qdb` file after export, you can view attributes of the database file in the Quartus Database File Viewer.

Table 13. Exporting Compilation Results

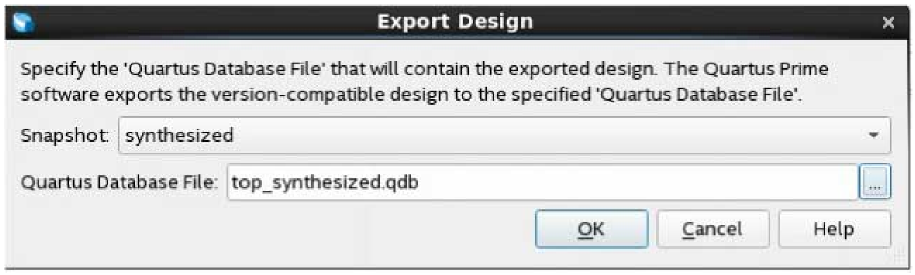
To Export Compilation Results For	Method	Description
Complete Design	Click **Project ➤ Export Design**	Saves compilation results for the entire project in a version-compatible Quartus database file (`.qdb`) that you can import to another project or migrate to a later version of the Intel Quartus Prime software. You can export the results for the synthesized or final compilation snapshot.
Design Partition	Click **Project ➤ Export Design Partition**	Saves compilation results for a design partition as a Partition Database File (`.qdb`) that you can import to another project using the same version of the Intel Quartus Prime software. You can export the results for the synthesized or final compilation snapshot.

1.6.1. Exporting a Version-Compatible Compilation Database

To export a project compilation database to a format that ensures version-compatibility with a later version of the Intel Quartus Prime software:

1. In the Intel Quartus Prime software, open the project that you want to export.
2. Generate synthesis or final compilation results by running one of the following commands:
   - Click **Processing ➤ Start ➤ Start Analysis & Synthesis** to generate synthesized compilation results.
   - Click **Processing ➤ Start Compilation** to generate final compilation results.
3. Click **Project ➤ Export Design**. Select the **synthesized** or **final Snapshot**.

Figure 49. Export Design Dialog Box



Specify the 'Quartus Database File' that will contain the exported design. The Quartus Prime software exports the version-compatible design to the specified 'Quartus Database File'.

Snapshot: synthesized

Quartus Database File: top_synthesized.qdb

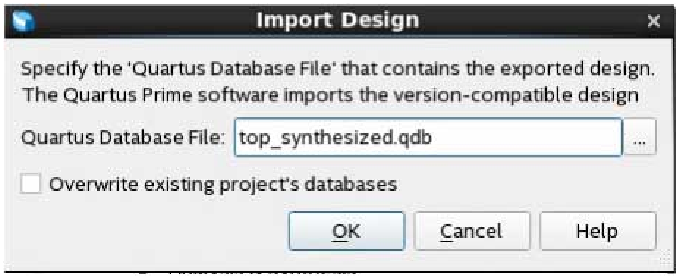
4. Specify a name for the **Quartus Database File** to contain the exported results, and click **OK**.
5. To include the exported design's settings and constraint files, copy the `.qsf` and `.sdc` files to the import project directory.

1.6.2. Importing a Version-Compatible Compilation Database

Follow these steps to import a project compilation database into a newer version of the Intel Quartus Prime software:

1. Export a version-compatible compilation database for a complete design, as Exporting a Version-Compatible Compilation Database on page 50 describes.
2. In a newer version of the Intel Quartus Prime software, open the original project. Click **Yes** if prompted to open a project created with a different software version.
3. Click **Project ➤ Import Design** and specify the **Quartus Database File**. To remove previous results, turn on **Overwrite existing project's databases**

Figure 50. Import Design Dialog Box



Specify the 'Quartus Database File' that contains the exported design. The Quartus Prime software imports the version-compatible design

Quartus Database File: top_synthesized.qdb

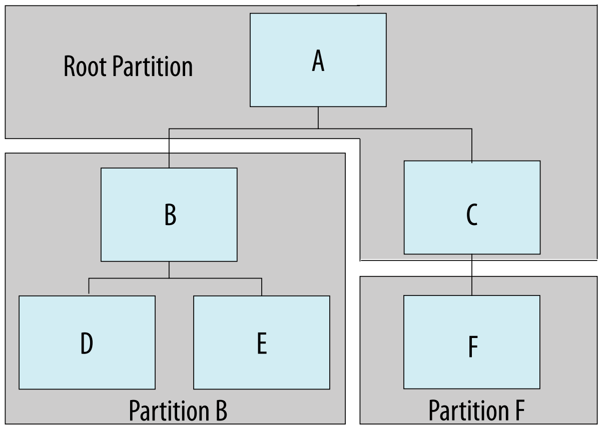
Overwrite existing project's databases

4. Click **OK**.
5. When you compile the imported design, run only Compiler stages that occur after the stage the `.qdb` preserves, rather than running a full compilation. For example, if you import a version-compatible database that contains the synthesis snapshot, start compilation with the Fitter (**Processing ➤ Start ➤ Start Fitter**). If you import a version-compatible database the contains the final snapshot, start compilation with Timing Analysis (Signoff) (**Processing ➤ Start ➤ Start Timing Analysis (Signoff)**).

1.6.3. Creating a Design Partition

A design partition is a logical, named, hierarchical boundary that you can assign to an instance in your design. Defining a design partition allows you to optimize and lock down the compilation results for individual blocks. You can then optionally export the compilation results of a design partition for reuse in another context, such as reuse in another project.

Figure 51. Design Partitions in Design Hierarchy



Follow these steps to create and modify design partitions:

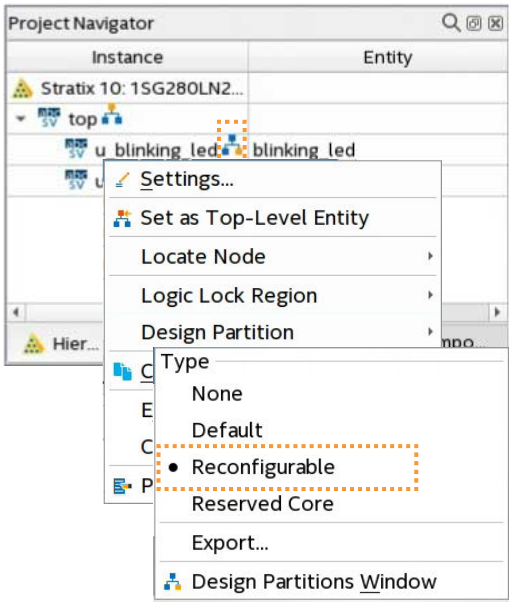
1. In the Intel Quartus Prime software, open the project that you want to partition.
2. Generate synthesis or final compilation results by running one of the following commands:

Send Feedback

- Click **Processing ➤ Start ➤ Start Analysis & Synthesis** to generate synthesized compilation results.
- Click **Processing ➤ Start Compilation** to generate final compilation results.

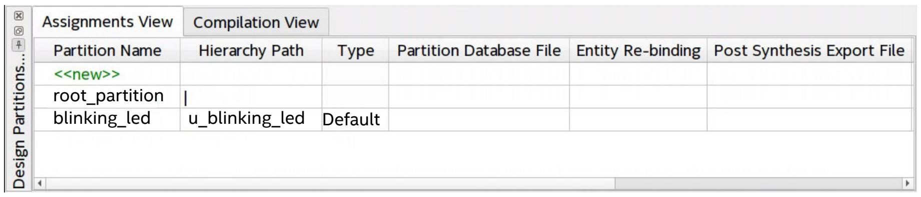
3. In the Project Navigator, right-click an instance in the **Hierarchy** tab, click **Design Partition ➤ Set as Design Partition**.

Figure 52. Creating a Design Partition from the Project Hierarchy



4. To view and edit all design partitions in the project, click **Assignments ➤ Design Partitions Window**.

Figure 53. Design Partitions Window



Partition Name	Hierarchy Path	Type	Partition Database File	Entity Re-binding	Post Synthesis Export File
<<new>>					
root_partition	|				
blinking_led	u_blinking_led	Default			

5. Specify the properties of the design partition in the Design Partitions Window. The following settings are available:

Table 14. Design Partition Settings

Option	Description
Partition Name	Specifies the partition name. Each partition name must be unique and consist of only alphanumeric characters. The Intel Quartus Prime software automatically creates a top-level (|) "root_partition" for each project revision.
Hierarchy Path	Specifies the hierarchy path of the entity instance that you assign to the partition. You specify this value in the **Create New Partition** dialog box. The root partition hierarchy path is |.
Type	Double-click to specify one of the following partition types that control how the Compiler processes and implements the partition:

continued...

Option	Description
	• **Default**—Identifies a standard partition. The Compiler processes the partition using the associated design source files. • **Reconfigurable**—Identifies a reconfigurable partition in a partial reconfiguration flow. Specify the **Reconfigurable** type to preserve synthesis results, while allowing refit of the partition in the PR flow. • **Reserved Core**—Identifies a partition in a block-based design flow that is reserved for core development by a Consumer reusing the device periphery.
Preservation Level	Specifies one of the following preservation levels for the partition: • **Not Set**—specifies no preservation level. The partition compiles from source files. • **synthesized**—the partition compiles using the synthesized snapshot. • **final**—the partition compiles using the final snapshot.
Empty	Specifies an empty partition that the Compiler skips. This setting is incompatible with the **Reserved Core** and **Partition Database File** settings for the same partition. The **Preservation Level** must be **Not Set**. An empty partition cannot have any child partitions.
Partition Database File	Specifies a Partition Database File (`.qdb`) that the Compiler uses during compilation of the partition. You export the `.qdb` for the stage of compilation that you want to reuse (synthesized or final). Assign the `.qdb` to a partition to reuse those results in another context.
Entity Re-binding	• PR Flow—specifies the entity that replaces the default persona in each implementation revision. • Root Partition Reuse Flow —specifies the entity that replaces the reserved core logic in the consumer project.
Color	Specifies the color-coding of the partition in the Chip Planner and Design Partition Planner displays.
Post Synthesis Export File	Automatically exports post-synthesis compilation results for the partition to the `.qdb` that you specify, each time Analysis & Synthesis runs. You can automatically export any design partition that does not have a preserved parent partition, including the root_partition.
Post Final Export File	Automatically exports post-final compilation results for the partition to the `.qdb` that you specify, each time the final stage of the Fitter runs. You can automatically export any design partition that does not have a preserved parent partition, including the root_partition.

1.6.4. Exporting a Design Partition

The following steps describe export of design partitions that you create in your project.

When you compile a design containing design partitions, the Compiler can preserve a synthesis or final snapshot of results for each partition. You can export the synthesized or final compilation results for individual design partitions with the **Export Design Partition** dialog box.

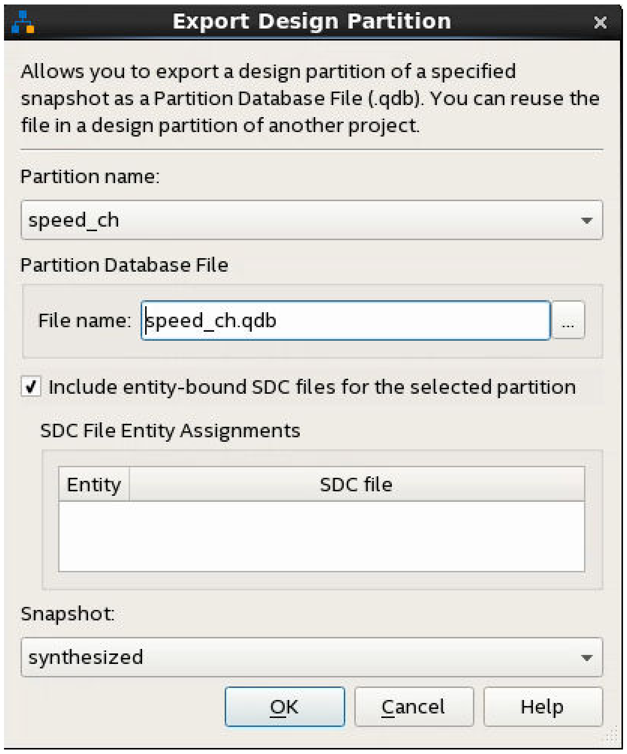
If the partition includes any entity-bound `.sdc` files, you can include those constraints in the `.qdb`. In addition, you can automate export of one or more partitions in the Design Partitions Window.

Manual Design Partition Export

Follow these steps to manually export a design partition with the **Export Design Partition** dialog box:

1. Open a project and create one or more design partitions. Creating a Design Partition on page 52 describes this process.
2. Run synthesis (**Processing ➤ Start ➤ Start Analysis & Synthesis**) or full compilation (**Processing ➤ Start Compilation**), depending on which compilation results that you want to export.
3. Click **Project ➤ Export Design Partition**, and specify one or more options in the **Export Design Partition** dialog box:

Figure 54. Export Design Partition Dialog Box



Allows you to export a design partition of a specified snapshot as a Partition Database File (.qdb). You can reuse the file in a design partition of another project.

Partition name: speed_ch

Partition Database File

File name: speed_ch.qdb

✓ Include entity-bound SDC files for the selected partition

SDC File Entity Assignments

Entity	SDC file

Snapshot: synthesized

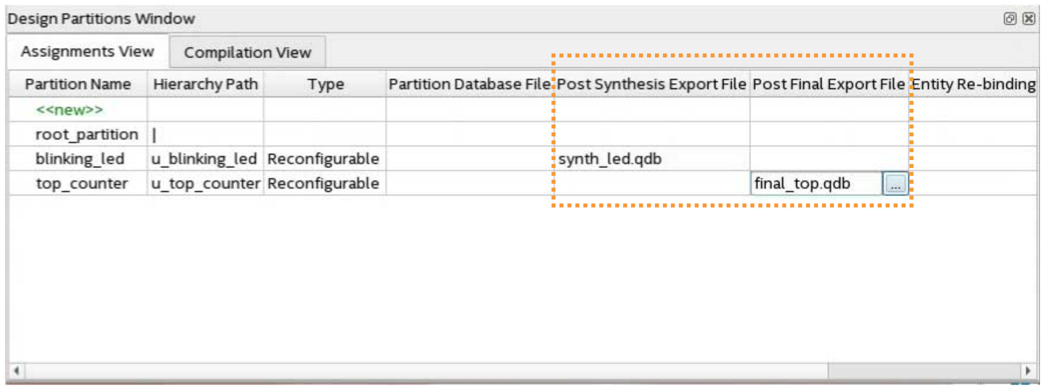
   - Select the **Partition name** and the compilation **Snapshot** for export.
   - To include any entity-bound `.sdc` files in the exported `.qdb`, turn on **Include entity-bound SDC files for the selected partition**.
4. Click **OK**. The compilation results for the design partition exports to the file that you specify.

Automated Design Partition Export

Follow these steps to automatically export one or more design partitions following each compilation:

1. Open a project containing one or more design partitions. Creating a Design Partition on page 52 describes this process.
2. To open the Design Partitions Window, click **Assignments ➤ Design Partitions Window**.
3. To automatically export a partition with synthesis results after each time you run synthesis, specify the a `.qdb` export path and file name for the **Post Synthesis Export File** option for that partition. If you specify only a file name without path, the file exports to the `output_files` directory after compilation.
4. To automatically export a partition with final snapshot results each time you run the Fitter, specify a `.qdb` file name for the **Post Final Export File** option for that partition. If you specify only a file name without path, the file exports to the `output_files` directory after compilation.

Figure 55. Specifying Export File in Design Partitions Window



Partition Name	Hierarchy Path	Type	Partition Database File	Post Synthesis Export File	Post Final Export File	Entity Re-binding
<<new>>						
root_partition	|					
blinking_led	u_blinking_led	Reconfigurable		synth_led.qdb		
top_counter	u_top_counter	Reconfigurable			final_top.qdb ...	

`.qsf` Equivalent Assignment:

```
set_instance_assignment -name EXPORT_PARTITION_SNAPSHOT_<FINAL|SYNTHESIZED> \
     <hpath> -to <file_name>.qdb
```

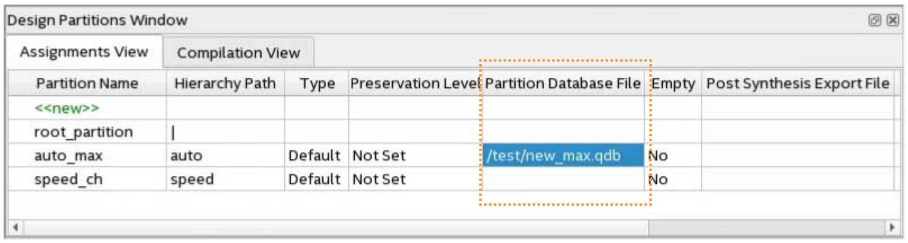
1.6.5. Reusing a Design Partition

You can reuse the compilation results of a design partition exported from another Intel Quartus Prime project. Reuse of a design partition allows you to share a synthesized or final design block with another designer.. Refer to *Intel Quartus Prime Pro Edition User Guide: Block-Based Design* for more information about reuse of design partitions.

To reuse an exported design partition in another project, you assign the exported partition `.qdb` to an appropriately configured design partition in the target project via the Design Partition Window:

1. Export a design partition with the appropriate snapshot, as Exporting a Design Partition on page 54 describes.
2. Open the target Intel Quartus Prime project that you want to reuse the exported partition.
3. Click **Processing ➤ Start ➤ Start Analysis & Elaboration**.
4. Click **Assignments ➤ Design Partitions Window**, and then create an appropriately sized design partition to contain the logic and compilation results of the exported `.qdb`.
5. Click the **Partition Database File** option for the new partition and select the exported `.qdb` file.

Figure 56. Partition Database File Setting in Design Partitions Window



Partition Name	Hierarchy Path	Type	Preservation Level	Partition Database File	Empty	Post Synthesis Export File
<<new>>						
root_partition	|					
auto_max	auto	Default	Not Set	/test/new_max.qdb	No	
speed_ch	speed	Default	Not Set		No	

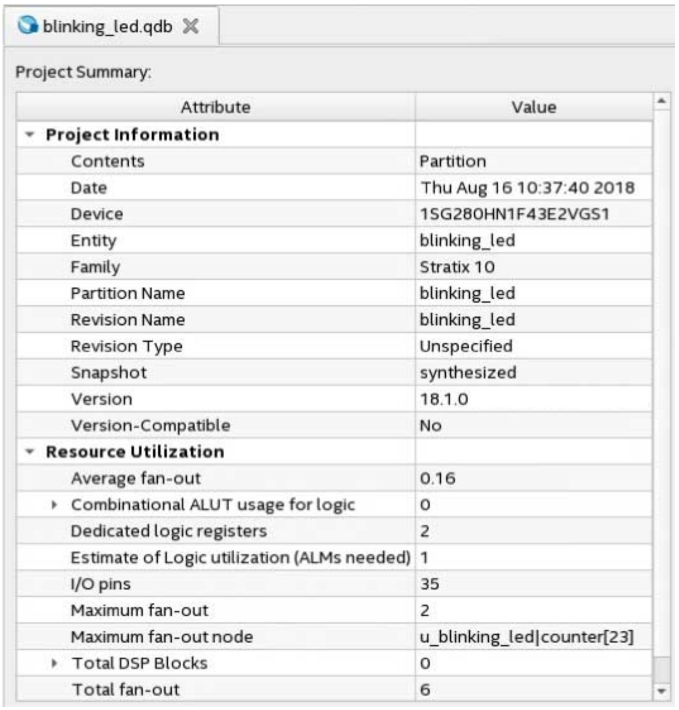
6. Specify any other properties of the design partition in the Design Partitions Window. The Compiler uses the partition's assigned `.qdb` as the source.

1.6.6. Viewing Quartus Database File Information

Although you cannot directly read a `.qdb` file, you can view helpful attributes about the file to quickly identify its contents and suitability for use.

The Intel Quartus Prime software automatically stores metadata about the project of origin when you export a Quartus Database File (`.qdb`). The Intel Quartus Prime software automatically stores metadata about the project of origin and resource utilization when you export a Partition Database File (`.qdb`) from your project. You can then use the Quartus Database File Viewer to display the attributes any of these `.qdb` files.

Figure 57. Quartus Database File Viewer



Project Summary:

Attribute	Value
Project Information	
Contents	Partition
Date	Thu Aug 16 10:37:40 2018
Device	1SG280HN1F43E2VGS1
Entity	blinking_led
Family	Stratix 10
Partition Name	blinking_led
Revision Name	blinking_led
Revision Type	Unspecified
Snapshot	synthesized
Version	18.1.0
Version-Compatible	No
Resource Utilization	
Average fan-out	0.16
Combinational ALUT usage for logic	0
Dedicated logic registers	2
Estimate of Logic utilization (ALMs needed)	1
I/O pins	35
Maximum fan-out	2
Maximum fan-out node	u_blinking_led|counter[23]
Total DSP Blocks	0
Total fan-out	6

Follow these steps to view the attributes of a .qdb file:

1. In the Intel Quartus Prime software, click **File ➤ Open**, select **Design Files** for **Files of Type**, and select a .qdb file.
2. Click **Open**. The Quartus Database File Viewer displays project and resource utilization attributes of the .qdb.

   Alternatively, run the following command-line equivalent:

```
quartus_cdb --extract_metadata --file <archive_name.qdb> \
     --type quartus --dir <extraction_directory> \
     [--overwrite]
```

1.6.6.1. QDB File Attribute Types

The Quartus Database Viewer can display the following attributes of a .qdb file:

Table 15. QDB File Attributes

QDB Attribute Types	Attribute	Example
Project Information	Contents	`Partition`
	Date	`Thu Jan 23 10:56:23 2018`
	Device	`10AX016C3U19E2LG`
	Entity (if Partition)	`Counter`
	Family	`Arria 10`
	Partition Name	`root_partition`

continued...

Send Feedback

	Revision Name	`Top`
	Revision Type	PR_BASE
	Snapshot	`synthesized`
	Version	`18.1.0 Pro Edition`
	Version-Compatible	`Yes`

Resource Utilization (exported for partition QDB only)	For synthesized snapshot partition lists data from the **Synthesis Resource Usage Summary** report.	`Average fan-out.16`
		`Dedicated logic registers:14`
		`Estimate of Logic utilization:1`
		`I/O pins:35`
		`Maximum fan-out:2`
		`Maximum fan-out node:counter[23]`
		`Total DSP Blocks:0`
		`Total fan-out:6`
		...

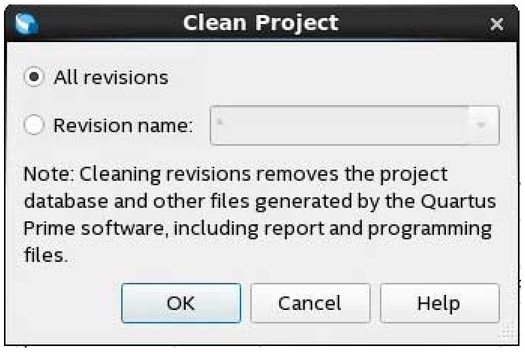
	For the final snapshot partition, lists data from the **Fitter Partition Statistics** report.	`Average fan-out:.16`
		`Combinational ALUTs: 16`
		`I/O Registers`
		`M20Ks`
		...

1.6.7. Clearing Compilation Results

You can clean the project database if you want to remove prior compilation results for all project revisions or for specific revisions. For example, you must clear previous compilation results before importing a version-compatible database to an existing project.

1. Click **Project > Clean Project**.
2. Select **All revisions** to clear the databases for all revisions of the current project, or specify a **Revision name** to clear only the revision's database you specify.
3. Click **OK**. A message indicates when the database is clean.

Figure 58. Clean Project Dialog Box Cleans the Project Database



[✓ All revisions]
Revision name:

Note: Cleaning revisions removes the project database and other files generated by the Quartus Prime software, including report and programming files.

1.7. Reducing Compilation Time

The Intel Quartus Prime Pro Edition software supports various strategies to reduce overall design compilation time. Running a full compilation including all Compiler modules on a large design can be time consuming. Use any the following techniques to reduce the overall compilation times of your design:

- Parallel compilation—the Compiler detects and uses multiple processors to reduce compilation time (for systems with multiple processor cores).
- Incremental optimization—breaks compilation into separate stages, allowing iterative analysis of results and optimization of settings at various compilation stages, prior to running a full compilation.
- Rapid Recompile of changed blocks—the Compiler reuses previous compilation results and does not reprocess unchanged design blocks. This flow is recommended for making small design changes in HDL or for adding or changing Signal Tap debug logic.

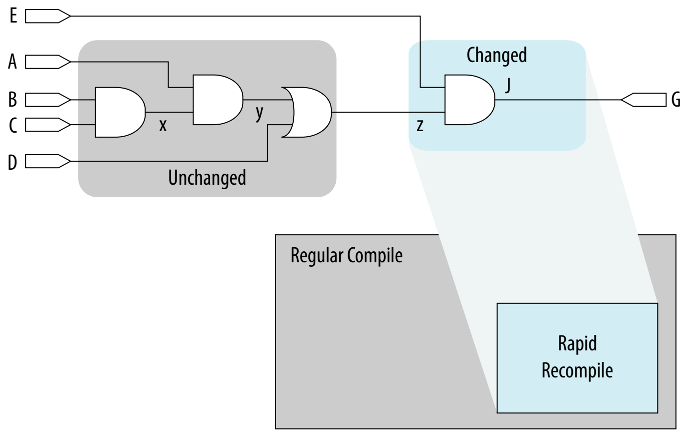
Related Information

- Running Rapid Recompile on page 60
- Reducing Compilation Time on page 87

1.7.1. Running Rapid Recompile

During Rapid Recompile the Compiler reuses previous synthesis and fitting results whenever possible, and does not reprocess unchanged design blocks. Use Rapid Recompile to reduce timing variations and the total recompilation time after making small design changes.

Figure 59. Rapid Recompile



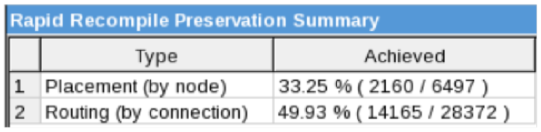
To run Rapid Recompile, follow these steps:

1. Prior to initial compilation, click **Assignments ➤ Settings ➤ Compiler Settings** and turn on **Enable Intermediate Fitter Snapshots**. This option must be enabled to subsequently use the Rapid Recompile feature.
2. To start Rapid Recompile following an initial compilation (or after running the Route stage of the Fitter), click **Processing ➤ Start ➤ Start Rapid Recompile**. Rapid Recompile implements the following types of design changes without full recompilation:

- Changes to nodes tapped by the Signal Tap Logic Analyzer
- Changes to combinational logic functions
- Changes to state machine logic (for example, new states, state transition changes)
- Changes to signal or bus latency or addition of pipeline registers
- Changes to coefficients of an adder or multiplier
- Changes to register packing behavior of DSP, RAM, or I/O
- Removal of unnecessary logic
- Changes to synthesis directives

3. Click the Rapid Recompile Preservation Summary report to view detailed information about the percentage of preserved compilation results.

Figure 60. Rapid Recompile Preservation Summary


Rapid Recompile Preservation Summary

	Type	Achieved
1	Placement (by node)	33.25 % (2160 / 6497)
2	Routing (by connection)	49.93 % (14165 / 28372)

1.7.2. Enabling Multi-Processor Compilation

The Compiler can detect and use multiple processors to reduce total compilation time. You specify the number of processors the Compiler uses. The Intel Quartus Prime software can use up to 16 processors to run algorithms in parallel. The Compiler uses parallel compilation by default. To reserve some processors for other tasks, specify a maximum number of processors that the software uses.

This technique reduces the compilation time by up to 10% on systems with two processing cores, and by up to 20% on systems with four cores. When running timing analysis independently, two processors reduce the timing analysis time by an average of 10%. This reduction reaches an average of 15% when using four processors.

The Intel Quartus Prime software does not necessarily use all the processors that you specify during a given compilation. Additionally, the software never uses more than the specified number of processors. This fact enables you to work on other tasks without slowing down your computer. The use of multiple processors does not affect the quality of the fit. For a given Fitter seed, and given **Maximum processors allowed** setting on a specific design, the fit is exactly the same and deterministic. This remains true, regardless of the target machine, and the number of available processors. Different **Maximum processors allowed** specifications produce different results of the same quality. The impact is similar to changing the Fitter seed setting.

To enable multiprocessor compilation, follow these steps:

1. Open or create an Intel Quartus Prime project.
2. Click **Assignments ➤ Settings ➤ Compilation Process Settings**.
3. Under **Parallel compilation**, specify options for the number of processors the Compiler uses.
4. View detailed information about processor use in the Parallel Compilation report following compilation.

To specify the number of processors for compilation at the command line, use the following Tcl command in your script:

```
set_global_assignment -name NUM_PARALLEL_PROCESSORS <value>
```

In this case, <*value*> is an integer from 1 to 16.

If you want the Intel Quartus Prime software to detect the number of processors and use all the processors for the compilation, include the following Tcl command in your script:

```
set_global_assignment -name NUM_PARALLEL_PROCESSORS ALL
```

Note: The Compiler detects Intel Hyper-Threading® Technology (Intel® HT Technology) as a single processor. If your system includes a single processor with Intel HT Technology, set the number of processors to one. Do not use the Intel HT Technology for Intel Quartus Prime compilations.

1.7.3. Factors Affecting Compilation Results

Almost any change to the following project settings, hardware, or software can impact the results from one compilation to the next.

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

1.8. Generating Programming Files

The Compiler's Assembler module generates files for device programming. Run the Assembler automatically as part of a full compilation, or run the Assembler module independently after design place and route. After running the Assembler, you can use the Programmer to download configuration data to a device. The Assembler generates one or more of the following files according to your specification in the **Device & Pin Options** dialog box.

Table 16. Assembler Generated Programming Files

Programming File	Description
SRAM Object Files (`.sof`)	A binary file containing the data for configuring all SRAM-based Intel FPGA devices.
Programmer Object Files (`.pof`)	A binary file containing the data for programming an EEPROM-based Intel configuration device. For example, the EPCS16 and EPCS64 devices, which configure SRAM-based Intel FPGA devices.

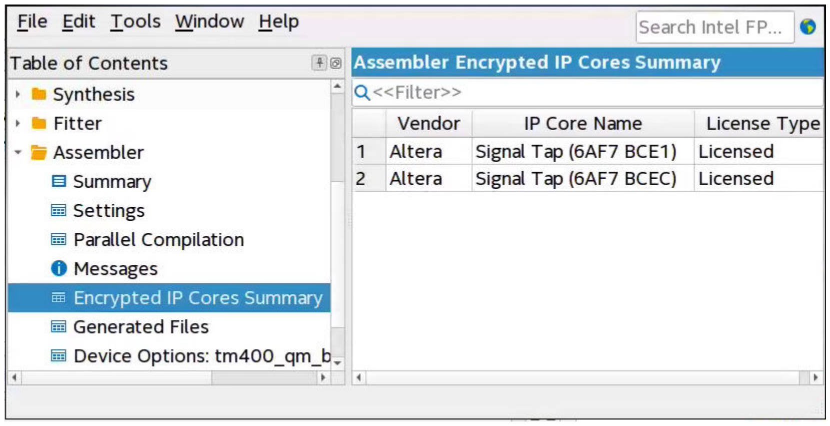
continued...

Programming File	Description
Hexadecimal (Intel-Format) Output Files (`.hexout`)	Contains configuration data that you can program into a parallel data source, such as an EPROM or a mass storage device, which configures an SRAM-based Intel FPGA device.
Raw Binary Files (`.rbf`)	Contains configuration data that an intelligent external controller uses to configure an SRAM-based Intel FPGA device.
Tabular Text Files (`.ttf`)	Contains configuration data that an intelligent external controller uses to configure an SRAM-based Intel FPGA device. *Note:* Generation of these files not available for Intel Stratix 10 designs.
Serial Vector Format File (.svf)	

1. Before running the Assembler, specify settings to customize programming file generation. Click **Assignments ➤ Device ➤ Device & Pin Options** to enable or disable generation of optional programming files.
2. To generate device programming files, click **Processing ➤ Start ➤ Start Assembler**, or click **Assembler** on the Compilation Dashboard. The Compiler confirms that prerequisite modules are complete, and launches the Assembler module to generate the programming files that you specify. The Messages window dynamically displays processing information, warnings, or errors. After Assembler processing,

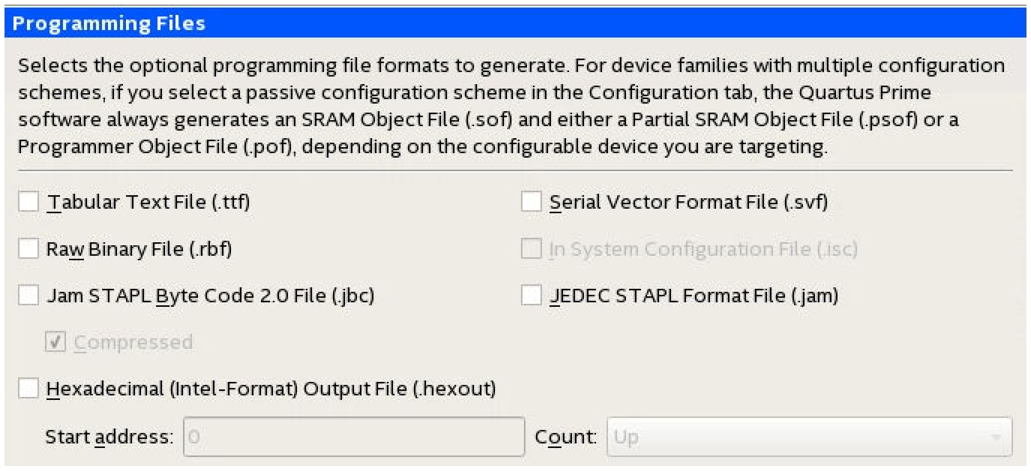
After running the Assembler, the Compilation report provides detailed information about programming file generation, including programming file Summary and Encrypted IP information.

Figure 61. Assembler Reports



	Vendor	IP Core Name	License Type
1	Altera	Signal Tap (6AF7 BCE1)	Licensed
2	Altera	Signal Tap (6AF7 BCEC)	Licensed

Figure 62. Device & Pin Options



Programming Files

Selects the optional programming file formats to generate. For device families with multiple configuration schemes, if you select a passive configuration scheme in the Configuration tab, the Quartus Prime software always generates an SRAM Object File (.sof) and either a Partial SRAM Object File (.psof) or a Programmer Object File (.pof), depending on the configurable device you are targeting.

- [ ] Tabular Text File (.ttf)
- [ ] Serial Vector Format File (.svf)
- [ ] Raw Binary File (.rbf)
- [ ] In System Configuration File (.isc)
- [ ] Jam STAPL Byte Code 2.0 File (.jbc)
- [ ] JEDEC STAPL Format File (.jam)
- [✓] Compressed
- [ ] Hexadecimal (Intel-Format) Output File (.hexout)

Start address: 0 Count: Up

Related Information

Intel Quartus Prime Pro Edition User Guide: Programmer

1.9. Synthesis Language Support

The Intel Quartus Prime software synthesizes standard Verilog HDL, VHDL, and SystemVerilog design files.

1.9.1. Verilog and SystemVerilog Synthesis Support

Intel Quartus Prime synthesis supports the following Verilog HDL language standards:

- Verilog-1995 (IEEE Standard 1364-1995)
- Verilog-2001 (IEEE Standard 1364-2001)
- SystemVerilog-2005 (IEEE Standard 1800-2005)
- SystemVerilog-2009 (IEEE Standard 1800-2009)

The following important guidelines apply to Intel Quartus Prime synthesis of Verilog HDL and SystemVerilog:

- The Compiler uses the Verilog-2001 standard by default for files with an extension of `.v`, and the SystemVerilog standard for files with the extension of `.sv`.
- If you use scripts to add design files, you can use the `-HDL_VERSION` command to specify the HDL version for each design file.
- Compiler support for Verilog HDL is case sensitive in accordance with the Verilog HDL standard.
- The Compiler supports the compiler directive `` `define``, in accordance with the Verilog HDL standard.

- The Compiler supports the `include` compiler directive to include files with absolute paths (with either "/" or "\" as the separator), or relative paths.
- When searching for a relative path, the Compiler initially searches relative to the project directory. If the Compiler cannot find the file, the Compiler next searches relative to all user libraries. Finally, the Compiler searches relative to the current file's directory location.
- Intel Quartus Prime Pro Edition synthesis searches for all modules or entities earlier in the synthesis process than other Quartus software tools. This earlier search produces earlier syntax errors for undefined entities than other Quartus software tools.

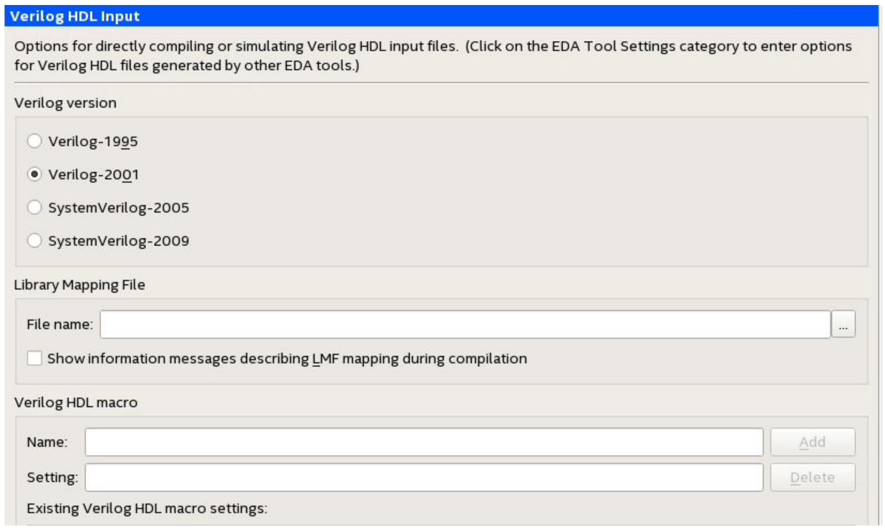
Related Information

- Intel Quartus Prime Pro Edition User Guide: Timing Analyzer
- Intel Quartus Prime Pro Edition User Guide: Design Recommendations

1.9.1.1. Verilog HDL Input Settings (Settings Dialog Box)

Click **Assignments ➤ Settings ➤ Verilog HDL Input** to specify options for the synthesis of Verilog HDL input files.

Figure 63. Verilog HDL Input Settings Dialog Box



Verilog HDL Input

Options for directly compiling or simulating Verilog HDL input files. (Click on the EDA Tool Settings category to enter options for Verilog HDL files generated by other EDA tools.)

Verilog version

- ○ Verilog-1995
- ● Verilog-2001
- ○ SystemVerilog-2005
- ○ SystemVerilog-2009

Library Mapping File

File name:

☐ Show information messages describing LMF mapping during compilation

Verilog HDL macro

Name:

Setting:

Existing Verilog HDL macro settings:

Table 17. Verilog HDL Input Settings

Setting	Description
Verilog Version	Directs synthesis to process Verilog HDL input design files using the specified standard. You can select any of the supported language standards to match your Verilog HDL files or SystemVerilog design files.
Library Mapping File	Allows you to optionally specify a provided Library Mapping File (`.lmf`) for use in synthesizing Verilog HDL files that contain non-Intel FPGA functions mapped to IP cores. You can specify the full path name of the LMF in the **File name** box.
Verilog HDL Macro	Verilog HDL macros are pre-compiler directives which can be added to Verilog HDL files to define constants, flags, or other features by **Name** and **Setting**. Macros that you add appear in the **Existing Verilog HDL macro settings** list.

1.9.1.2. Design Libraries

By default, the Compiler processes all design files into one or more libraries.

- When compiling a design instance, the Compiler initially searches for the entity in the library associated with the instance (which is the work library if you do not specify any library).
- If the Compiler cannot locate the entity definition, the Compiler searches for a unique entity definition in all design libraries.
- If the Compiler finds more than one entity with the same name, the Compiler generates an error. If your design uses multiple entities with the same name, you must compile the entities into separate libraries.

1.9.1.3. Verilog HDL Configuration

Verilog HDL configuration is a set of rules that specify the source code for particular instances. Verilog HDL configuration allows you to perform the following tasks:

- Specify a library search order for resolving cell instances (as does a library mapping file).
- Specify overrides to the logical library search order for specified instances.
- Specify overrides to the logical library search order for all instances of specified cells.

1.9.1.3.1. Hierarchical Design Configurations

A design can have more than one configuration. For example, you can define a configuration that specifies the source code you use in particular instances in a sub-hierarchy, and then define a configuration for a higher level of the design.

For example, suppose a subhierarchy of a design is an eight-bit adder, and the RTL Verilog code describes the adder in a logical library named `rtllib`. The gate-level code describes the adder in the `gatelib` logical library. If you want to use the gate-level code for the 0 (zero) bit of the adder and the RTL level code for the other seven bits, the configuration might appear as follows:

Example 1. Gate-level code for the 0 (zero) bit of the adder

```
config cfg1;
design aLib.eight_adder;
default liblist rtllib;
instance adder.fulladd0 liblist gatelib;
endconfig
```

If you are instantiating this eight-bit adder eight times to create a 64-bit adder, use configuration `cfg1` for the first instance of the eight-bit adder, but not in any other instance. A configuration that performs this function is shown below:

Example 2. Use configuration `cfg1` for first instance of eight-bit adder

```
config cfg2;
design bLib.64_adder;
default liblist bLib;
instance top.64add0 use work.cfg1:config;
endconfig
```

Note: The name of the unbound module may be different from the name of the cell that is bounded to the instance.

1.9.1.4. Initial Constructs and Memory System Tasks

The Intel Quartus Prime software infers power-up conditions from the Verilog HDL `initial` constructs. The Intel Quartus Prime software also creates power-up settings for variables, including RAM blocks. If the Intel Quartus Prime software encounters non-synthesizable constructs in an `initial` block, it generates an error.

To avoid such errors, enclose non-synthesizable constructs (such as those intended only for simulation) in `translate_off` and `translate_on` synthesis directives. Synthesis of initial constructs enables the power-up state of the synthesized design to match the power-up state of the original HDL code in simulation.

Note: Initial blocks do not infer power-up conditions in some third-party EDA synthesis tools. If you convert between synthesis tools, you must set your power-up conditions correctly.

Intel Quartus Prime synthesis supports the `$readmemb` and `$readmemh` system tasks to initialize memories.

Example 3. Verilog HDL Code: Initializing RAM with the readmemb Command

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

When creating a text file to use for memory initialization, specify the address using the format @<*location*> on a new line, and then specify the memory word such as `110101` or `abcde` on the next line.

The following example shows a portion of a Memory Initialization File (`.mif`) for the RAM.

Example 4. Text File Format: Initializing RAM with the readmemb Command

```
@0
00000000
@1
00000001
@2
00000010
…
@e
00001110
@f
00001111
```

1.9.1.5. Verilog HDL Macros

The Intel Quartus Prime software fully supports Verilog HDL macros, which you can define with the `'define` compiler directive in your source code. You can also define macros in the Intel Quartus Prime software or on the command line.

To set Verilog HDL macros at the command line for the Intel Quartus Prime Pro Edition synthesis (`quartus_syn`) executable, use the following format:

```
quartus_syn <PROJECT_NAME> --set=VERILOG_MACRO=a=2
```

This command adds the following new line to the project `.qsf` file:

```
set_global_assignment -name VERILOG_MACRO "a=2"
```

To avoid adding this line to the project `.qsf`, add this option to the `quartus_syn` command:

```
--write_settings_files=off
```

1.9.2. VHDL Synthesis Support

Intel Quartus Prime synthesis supports the following VHDL language standards.

- VHDL 1987 (IEEE Standard 1076-1987)
- VHDL 1993 (IEEE Standard 1076-1993)
- VHDL 2008 (IEEE Standard 1076-2008)

The Intel Quartus Prime Compiler uses the VHDL 1993 standard by default for files that have the extension `.vhdl` or `.vhd`.

Note: The VHDL code samples follow the VHDL 1993 standard.

Related Information

Migrating to Quartus Prime Pro Edition

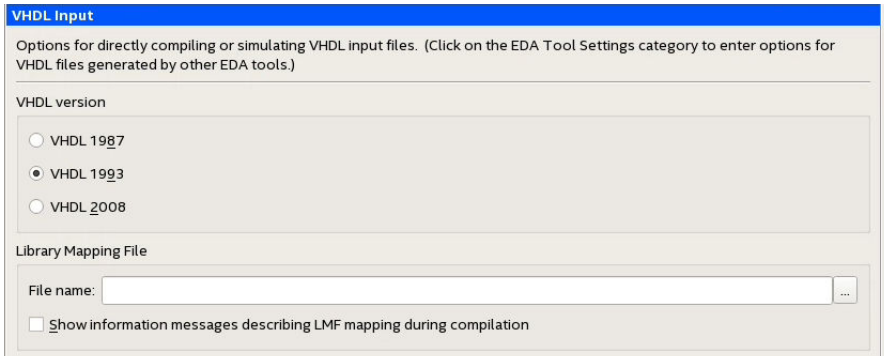
1.9.2.1. VHDL Input Settings (Settings Dialog Box)

Click **Assignments ➤ Settings ➤ VHDL Input** to specify options for the synthesis of VHDL input files.

Table 18. VHDL Input Settings

Setting	Description
VHDL Version	Specifies the VHDL standard for use during synthesis of VHDL input design files. Select the language standards that corresponds with the VHDL files.
Library Mapping File	Specifies a Library Mapping File (`.lmf`) for use in synthesizing VHDL files that contain IP cores. Specify the full path name of the LMF in the **File name** box.

Figure 64. VHDL Input Settings Dialog Box


VHDL Input

Options for directly compiling or simulating VHDL input files. (Click on the EDA Tool Settings category to enter options for VHDL files generated by other EDA tools.)

VHDL version

- ○ VHDL 1987
- ● VHDL 1993
- ○ VHDL 2008

Library Mapping File

File name:

☐ Show information messages describing LMF mapping during compilation

1.9.2.2. VHDL Standard Libraries and Packages

The Intel Quartus Prime software includes the standard IEEE libraries and several vendor-specific VHDL libraries. The IEEE library includes the standard VHDL packages `std_logic_1164`, `numeric_std`, `numeric_bit`, and `math_real`.

The STD library is part of the VHDL language standard and includes the packages `standard` (included in every project by default) and `textio`. For compatibility with older designs, the Intel Quartus Prime software also supports the following vendor-specific packages and libraries:

- Synopsys* packages such as `std_logic_arith` and `std_logic_unsigned` in the IEEE library.
- Mentor Graphics* packages such as `std_logic_arith` in the ARITHMETIC library.
- Primitive packages `altera_primitives_components` (for primitives such as GLOBAL and DFFE) and `maxplus2` in the ALTERA library.
- IP core packages `altera_mf_components` in the ALTERA_MF library for specific IP cores including LCELL. In addition, `lpm_components` in the LPM library for library of parameterized modules (LPM) functions.

Note: Import component declarations for primitives such as GLOBAL and DFFE from the `altera_primitives_components` package and not the `altera_mf_components` package.

1.9.2.3. VHDL wait Constructs

The Intel Quartus Prime software supports one VHDL `wait until` statement per process block. However, the Intel Quartus Prime software does not support other VHDL wait constructs, such as `wait for` and `wait on` statements, or processes with multiple `wait` statements.

Example 5. VHDL `wait until` construct example

```
architecture dff_arch of ls_dff is
begin
output: process begin
wait until (CLK'event and CLK='1');
Q <= D;
```

```
Qbar <= not D;
end process output;
end dff_arch;
```

1.10. Synthesis Settings Reference

This section provides a reference to all synthesis settings. Use these settings to customize synthesis processing for your design goals.

1.10.1. Enable Intermediate Fitter Snapshots

To save compilation time, the Compiler does not save the planned, placed, routed, or retimed snapshots by default during full compilation.

However, you can turn on **Enable Intermediate Fitter Snapshots** (**Assignments ➤ Settings ➤ Compiler Settings**) to generate and preserve snapshots for the Plan, Place, Route, and Retime stages any time you run full compilation. You can also run any intermediate Fitter stage independently to generate the snapshot for that stage.

Note: You must enable **Enable Intermediate Fitter Snapshots** to subsequently use the Rapid Recompile feature.

1.10.2. Optimization Modes

The following options direct the focus of Compiler optimization efforts during synthesis. Specify a **Balanced** strategy, or optimize for **Performance**, **Area**, **Routability**, **Power**, or **Compile Time**. The Compiler targets the optimization goal you specify. The settings affect synthesis and fitting.

Table 19. Optimization Modes (Compiler Settings Page)

Optimization Mode	Description
Balanced (normal flow)	The Compiler optimizes synthesis for balanced implementation that respects timing constraints.
High Performance Effort	The Compiler increases the timing optimization effort during placement and routing, and enables timing-related Physical Synthesis optimizations (per register optimization settings). Each additional optimization can increase compilation time.
High Performance with Maximum Placement Effort	Enables the same Compiler optimizations as **High Performance Effort**, with additional placement optimization effort.
Superior Performance	Enables the same Compiler optimizations as **High Performance Effort**, and adds more optimizations during Analysis & Synthesis to maximize design performance with a potential increase to logic area. If design utilization is already very high, this option may lead to difficulty in fitting, which can also negatively affect overall optimization quality.
Superior Performance with Maximum Placement Effort	Enables the same Compiler optimizations as **Superior Performance**, with additional placement optimization effort.
Aggressive Area	The Compiler makes aggressive effort to reduce the device area required to implement the design at the potential expense of design performance.
High Placement Routability Effort	The Compiler makes high effort to route the design at the potential expense of design area, performance, and compilation time. The Compiler spends additional time reducing routing utilization, which can improve routability and also saves dynamic power.

continued...

Optimization Mode	Description
High Packing Routability Effort	The Compiler makes high effort to route the design at the potential expense of design area, performance, and compilation time. The Compiler spends additional time packing registers, which can improve routability and also saves dynamic power.
Optimize Netlist for Routability	The Compiler implements netlist modifications to increase routability at the possible expense of performance.
High Power Effort	The Compiler makes high effort to optimize synthesis for low power. **High Power Effort** increases synthesis run time.
Aggressive Power	Makes aggressive effort to optimize synthesis for low power. The Compiler further reduces the routing usage of signals with the highest specified or estimated toggle rates, saving additional dynamic power but potentially affecting performance.
Aggressive Compile Time	Reduces the compile time required to implement the design with reduced effort and fewer performance optimizations. This option also disables some detailed reporting functions. *Note:* Turning on **Aggressive Compile Time** enables Intel Quartus Prime Settings File (`.qsf`) settings which cannot be overridden by other `.qsf` settings.

Note: If you enable extended optimization modes for Design Space Explorer II by use of `.qsf` assignments, and then subsequently open the **Compiler Settings** tab for that project revision, the **Compiler Settings** tab indicates that the extended optimization mode reverts to one of the **Compiler Settings** tab **Optimization Modes**.

1.10.3. Allow Register Retiming

The **Allow Register Retiming** option controls whether or not to globally disable retiming. When turned on, the Compiler automatically performs register retiming optimizations, moving registers through combinational logic. When turned off, the Compiler prevents any retiming optimizations on a global scale. Optionally, assign **Allow Register Retiming** to any design entity or instance for specific portions of the design. Click **Assignments ➤ Assignment Editor** to specify entity- and instance-level assignments, or use the following syntax to make the assignment in the `.qsf` directly.

Example 6. Disable register retiming for entity abc

```
set_global_assignment –name ALLOW_REGISTER_RETIMING ON

set_instance_assignment –name ALLOW_REGISTER_RETIMING OFF –to “abc|”

set_instance_assignment –name ALLOW_REGISTER_RETIMING ON –to “abc|def|”
```

Example 7. Disable register retiming for the whole design, except for registers in entity abc

```
set_global_assignment –name ALLOW_REGISTER_RETIMING OFF

set_instance_assignment –name ALLOW_REGISTER_RETIMING ON –to “abc|”

set_instance_assignment –name ALLOW_REGISTER_RETIMING OFF –to “abc|def|”
```

1.10.4. Advanced Synthesis Settings

The following section is a quick reference of all Advanced Synthesis Settings. Click **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Synthesis)** to modify these settings.

Table 20. Advanced Synthesis Settings (1 of 13)

Option	Description
Allow Any RAM Size for Recognition	Allows the Compiler to infer RAMs of any size, even if the RAMs do not meet the current minimum requirements.
Allow Any ROM Size for Recognition	Allows the Compiler to infer ROMs of any size even if the ROMs do not meet the design's current minimum size requirements.
Allow Any Shift Register Size for Recognition	Allows the Compiler to infer shift registers of any size even if they do not meet the design's current minimum size requirements.
Allow Register Duplication	Controls whether the Compiler duplicates registers to improve design performance. When enabled, the Compiler performs optimization that creates a second copy of a register and move a portion of its fan-out to this new node. This technique improves routability and reduces the total routing wire required to route a net with many fan-outs. If you disable this option, retiming of registers is also disabled. *Note:* Not available for Intel Stratix 10 devices.
Allow Register Merging	Controls whether the Compiler removes (merges) identical registers. When enabled, in cases where two registers generate the same logic, the Compiler may delete one register and fan-out the remaining register to the deleted register's destinations. This option is useful if you want to prevent the Compiler from removing duplicate registers that you have used deliberately. When disabled, retiming optimizations are also disabled. *Note:* Not available for Intel Stratix 10 devices.
Allow Shift Register Merging Across Hierarchies	Allows the Compiler to take shift registers from different hierarchies of the design and put the registers in the same RAM.
Allow Synchronous Control Signals	Allows the Compiler to utilize synchronous clear and synchronous load signals in normal mode logic cells. Turning on this option helps to reduce the total number of logic cells used in the design, but can negatively impact the fitting. This negative impact occurs because all the logic cells in a LAB share synchronous control signals.

Table 21. Advanced Synthesis Settings (2 of 13)

Option	Description
Analysis & Synthesis Message Level	Specifies the type of Analysis & Synthesis messages the Compiler display. **Low** displays only the most important Analysis & Synthesis messages. **Medium** displays most messages, but hides the detailed messages. **High** displays all messages.
Auto Clock Enable Replacement	Allows the Compiler to locate logic that feeds a register and move the logic to the register's clock enable input port.
Auto DSP Block Replacement	Allows the Compiler to find a multiply-accumulate function or a multiply-add function that can be replaced with a DSP block.
Auto Gated Clock Conversion	Automatically converts gated clocks to use clock enable pins. Clock gating logic can contain AND, OR, MUX, and NOT gates. Turning on this option may increase memory use and overall run time. You must use the Timing Analyzer for timing analysis, and you must define all base clocks in Synopsys Design Constraints (`.sdc`) format.

Table 22. Advanced Synthesis Settings (3 of 13)

Option	Description
Auto Open-Drain Pins	Allows the Compiler to automatically convert a tri-state buffer with a strong low data input into the equivalent open-drain buffer.
Auto RAM Replacement	Allows the Compiler to identify sets of registers and logic that it can replace with the altsyncram or the lpm_ram_dp IP core. Turning on this option may change the functionality of the design.
Auto ROM Replacement	Allows the Compiler to identify logic that it can replace with the altsyncram or the lpm_rom IP core. Turning on this option may change the power-up state of the design.

continued...

Option	Description
Auto Resource Sharing	Allows the Compiler to share hardware resources among many similar, but mutually exclusive, operations in your HDL source code. If you enable this option, the Compiler merges compatible addition, subtraction, and multiplication operations. Merging operations may reduce the area your design requires. Because resource sharing introduces extra muxing and control logic on each shared resource, it may negatively impact the final f_{MAX} of your design.
Auto Shift Register Placement	Allows the Compiler to find a group of shift registers of the same length that are replaceable with the altshift_taps IP core. The shift registers must all use the same clock and clock enable signals. The registers must not have any other secondary signals. The registers must have equally spaced taps that are at least three registers apart.
Automatic Parallel Synthesis	Option to enable/disable automatic parallel synthesis. Use this option to speed up synthesis compile time by using multiple processors when available.

Table 23. Advanced Synthesis Settings (4 of 13)

Option	Description
Block Design Naming	Specifies the naming scheme for the block design. The Compiler ignores the option if you assign the option to anything other than a design entity.
Clock MUX Protection	Causes the multiplexers in the clock network to decompose to 2-to-1 multiplexer trees. The Compiler protects these trees from merging with, or transferring to, other logic. This option helps the Timing Analyzer to analyze clock behavior.
DSP Block Balancing	Allows you to control the conversion of certain DSP block slices during DSP block balancing.

Table 24. Advanced Synthesis Settings (5 of 13)

Option	Description
Disable DSP Negate Inferencing	Allows you to specify whether to use the negate port on an inferred DSP block.
Disable Register Merging Across Hierarchies	Specifies whether the Compiler allows merging of registers that are in different hierarchies if their inputs are the same.
Enable Formal Verification Support	Enables the Compiler to write scripts for use with the OneSpin* formal verification tool.
Enable State Machines Inference	Allows the Compiler to infer state machines from VHDL or Verilog HDL design files. The Compiler optimizes state machines to reduce area and improve performance. If set to **Off**, the Compiler extracts and optimizes state machines in VHDL or Verilog HDL design files as regular logic.
Force Use of Synchronous Clear Signals	Forces the Compiler to utilize synchronous clear signals in normal mode logic cells. Enabling this option helps to reduce the total number of logic cells in the design, but can negatively impact the fitting. All the logic cells in a LAB share synchronous control signals.
Fractal Synthesis	Turning this option **On** directs the Compiler to apply dense packing to arithmetic blocks, minimizing the area of the design for arithmetic-intensive designs.
HDL Message Level	Specifies the type of HDL messages you want to view, including messages that display processing errors in the HDL source code. **Level1** displays only the most important HDL messages. **Level2** displays most HDL messages, including warning and information based messages. **Level3** displays all HDL messages, including warning and information based messages and alerts about potential design problems or lint errors.

Table 25. Advanced Synthesis Settings (6 of 13)

Option	Description
Ignore GLOBAL Buffers	Ignores `GLOBAL` buffers in the design. The Compiler ignores this option if you apply the option to anything other than an individual `GLOBAL` buffer, or a design entity containing `GLOBAL` buffers.
Ignore LCELL Buffers	Ignores `LCELL` buffers in the design. The Compiler ignores this option if you apply the option to anything other than an individual `LCELL` buffer, or a design entity containing `LCELL` buffers.
Ignore Maximum Fan-Out Assignments	Directs the Compiler to ignore the Maximum Fan-Out Assignments on a node, an entity, or the whole design.
Ignore SOFT Buffers	Ignores `SOFT` buffers in the design. The Compiler ignores this option if you apply the option to anything other than an individual `SOFT` buffer or a design entity containing `SOFT` buffers.

Table 26. Advanced Synthesis Settings (7 of 13)

Option	Description
Ignore translate_off and synthesis_off Directives	Ignores all `translate_off/synthesis_off` synthesis directives in Verilog HDL and VHDL design files. Use this option to disable these synthesis directives and include previously ignored code during elaboration.
Infer RAMs from Raw Logic	Infers RAM from registers and multiplexers. The Compiler initially converts some HDL patterns differing from RAM templates into logic. However, these structures function as RAM. As a result, when you enable this option, the Compiler may substitute the altsyncram IP core instance for them at a later stage. When you enable this assignment, the Compiler may use more device RAM resources and fewer LABs.
Iteration Limit for Constant Verilog Loops	Defines the iteration limit for Verilog loops with loop conditions that evaluate to compile-time constants on each loop iteration. This limit exists primarily to identify potential infinite loops before they exhaust memory or trap the software in an actual infinite loop.
Iteration Limit for non-Constant Verilog Loops	Defines the iteration limit for Verilog HDL loops with loop conditions that do not evaluate to compile-time constants on each loop iteration. This limit exists primarily to identify potential infinite loops before they exhaust memory or trap the software in an actual infinite loop.

Table 27. Advanced Synthesis Settings (8 of 13)

Option	Description
Maximum DSP Block Usage	Specifies the maximum number of DSP blocks that the DSP block balancer assumes exist in the current device for each partition. This option overrides the usual method of using the maximum number of DSP blocks the current device supports.
Maximum Number of LABs	Specifies the maximum number of LABs that Analysis & Synthesis should try to utilize for a device. This option overrides the usual method of using the maximum number of LABs the current device supports, when the value is non-negative and is less than the maximum number of LABs available on the current device.
Maximum Number of M4K/M9K/M20K/M10K Memory Blocks	Specifies the maximum number of M4K, M9K, M20K, or M10K memory blocks that the Compiler may use for a device. This option overrides the usual method of using the maximum number of M4K, M9K, M20K, or M10K memory blocks the current device supports, when the value is non-negative and is less than the maximum number of M4K, M9K, M20K, or M10K memory blocks available on the current device.

Table 28. Advanced Synthesis Settings (9 of 13)

Option	Description
Maximum Number of Registers Created from Uninferred RAMs	Specifies the maximum number of registers that Analysis & Synthesis uses for conversion of uninferred RAMs. Use this option as a project-wide option or on a specific partition by setting the assignment on the instance name of the partition root. The

continued...

Option	Description
	assignment on a partition overrides the global assignment (if any) for that particular partition. This option prevents synthesis from causing long compilations and running out of memory when many registers are used for uninferred RAMs. Instead of continuing the compilation, the Intel Quartus Prime software issues an error and exits.
NOT Gate Push-Back	Allows the Compiler to push an inversion (that is, a NOT gate) back through a register and implement it on that register's data input if it is necessary to implement the design. When this option is on, a register may power-up to an active-high state, and may need explicit clear during initial operation of the device. The Compiler ignores this option if you apply it to anything other than an individual register or a design entity containing registers. When you apply this option to an output pin that is directly fed by a register, the assignment automatically transfers to that register.
Number of Inverted Registers Reported in Synthesis Report	Specifies the maximum number of inverted registers that the Synthesis report displays.
Number of Protected Registers Reported in Synthesis Report	Specifies the maximum number of protected registers that the Synthesis Report displays.
Number of Removed Registers Reported in Synthesis Migration Checks	Specifies the maximum number of rows that the Synthesis Migration Check report displays.
Number of Swept Nodes Reported in Synthesis Report	Specifies the maximum number of swept nodes that the Synthesis Report displays. A swept node is any node which was eliminated from your design because the Compiler found the node to be unnecessary.
Number of Rows Reported in Synthesis Report	Specifies the maximum number of rows that the Synthesis report displays. *Note:* Not available for Intel Stratix 10 devices.
Optimization Technique	Specifies an overall optimization goal for Analysis & Synthesis. Specify a **Balanced** strategy, or optimize for **Performance**, **Area**, **Routability**, **Power**, or **Compile Time**. The Compiler targets the optimization goal you specify.

Table 29. Advanced Synthesis Settings (10 of 13)

Option	Description
Perform WYSIWYG Primitive Resynthesis	Specifies whether to perform WYSIWYG primitive resynthesis during synthesis. This option uses the setting specified in the **Optimization Technique** logic option.
Power-Up Don't Care	Causes registers that do not have a **Power-Up Level logic** option setting to power-up with a do not care logic level (X). When the **Power-Up Don't Care** option is on, the Compiler determines when it is beneficial to change the power-up level of a register to minimize the area of the design. The Compiler maintains a power-up state of zero, unless there is an immediate area advantage.
Power Optimization During Synthesis	Controls the power-driven compilation setting of Analysis & Synthesis. This option determines how aggressively Analysis & Synthesis optimizes the design for power. When this option is **Off**, the Compiler does not perform any power optimizations. **Normal compilation** performs power optimizations provided that they are not expected to reduce design performance. **Extra effort** performs additional power optimizations which may reduce design performance.

Table 30. Advanced Synthesis Settings (11 of 13)

Option	Description
Remove Duplicate Registers	Removes a register if it is identical to another register. If two registers generate the same logic, the Compiler deletes the duplicate. The first instance fans-out to the duplicates destinations. Also, if the deleted register contains different logic option assignments, the Compiler ignores the options. This option is useful if you want to

continued...

Option	Description
	prevent the Compiler from removing intentionally duplicate registers. The Compiler ignores this option if you apply it to anything other than an individual register or a design entity containing registers.
Remove Redundant Logic Cells	Removes redundant LCELL primitives or WYSIWYG primitives. Turning this option on optimizes a circuit for area and speed. The Compiler ignores this option if you apply it to anything other than a design entity.
Report Parameter Settings	Specifies whether the Synthesis report includes the reports in the **Parameter Settings by Entity Instance** folder.
Report PR Initial Values as Errors	Allows you to flag explicitly defined initial values found in PR partitions as Errors instead of Warnings.
Report Source Assignments	Specifies whether the Synthesis report includes reports in the **Source Assignments** folder.

Table 31. Advanced Synthesis Settings (12 of 13)

Option	Description
Resource Aware Inference for Block RAM	Specifies whether RAM, ROM, and shift-register inference should take the design and device resources into account.
Restructure Multiplexers	Reduces the number of logic elements synthesis requires to implement multiplexers in a design. This option is useful if your design contains buses of fragmented multiplexers. This option repacks multiplexers more efficiently for area, allowing the design to implement multiplexers with a reduced number of logic elements: • **On**—minimizes your design area, but may negatively affect design clock speed (f_{MAX}). • **Off**—disables multiplexer restructuring; it does not decrease logic element usage and does not affect design clock speed (f_{MAX}). • **Auto**—allows the Intel Quartus Prime software to determine whether multiplexer restructuring should be enabled. The **Auto** setting decreases logic element usage, but may negatively affect design clock speed (f_{MAX}).
SDC Constraint Protection	Verifies `.sdc` constraints in register merging. This option helps to maintain the validity of `.sdc` constraints through compilation.
Safe State Machine	The **Safe State Machine** option implements state machines that can recover from an illegal state. The following settings are available: • **Auto**—for Intel Stratix 10 designs, this default setting enables **Safe State Machine** whenever the Compiler determines this setting is advantageous in state machines of 6 or less states. The setting helps to allow for unexpected initial power-up conditions. For Intel Arria 10 and Intel Cyclone 10 GX, the **Auto** setting is the same as **Never**. • **On**—directs the Compiler to always use **Safe State Machine**. • **Never**—never uses **Safe State Machine**.
Shift Register Replacement – Allow Asynchronous Clear Signal	Allows the Compiler to find a group of shift registers of the same length that can be replaced with the altshift_taps IP core. The shift registers must all use the same `aclr` signals, must not have any other secondary signals, and must have equally spaced taps that are at least three registers apart. To use this option, you must turn on the **Auto Shift Register Replacement** logic option.
Size of the Latch Report	Allows you to specify the maximum number of latches that the Synthesis Report should display.
Size of the PR Initial Conditions Report	Allows you to specify the maximum number of registers that the PR Initial Conditions Report should display.

Table 32. Advanced Synthesis Settings (13 of 13)

Option	Description
State Machine Processing	Specifies the processing style the Compiler uses to process a state machine. You can use your own **User-Encoded** style, or select **One-Hot**, **Minimal Bits**, **Gray**, **Johnson**, **Sequential,** or **Auto** (Compiler-selected) encoding.
Strict RAM Replacement	When this option is **On**, the Compiler replace RAM only if the hardware matches the design exactly.
Synchronization Register Chain Length	Specifies the maximum number of registers in a row that the Compiler considers as a synchronization chain. Synchronization chains are sequences of registers with the same clock and no fan-out in between, such that the first register is fed by a pin, or by logic in another clock domain. The Compiler considers these registers for metastability analysis. The Compiler prevents optimizations of these registers, such as retiming. When gate-level retiming is enabled, the Compiler does not remove these registers. The default length is set to two.
Synthesis Effort	Controls the synthesis trade-off between compilation speed, performance, and area. The default is **Auto**. You can select **Fast** for faster compilation speed at the cost of performance and area.
Synthesis Migration Check for Stratix 10	Enables synthesis checks on Intel Arria 10 to Intel Stratix 10 design migration.
Timing-Driven Synthesis	For Intel Arria 10 and Intel Cyclone 10 GX designs, allows synthesis to use timing information to better optimize the design. The **Timing-Driven Synthesis** logic option impacts the following **Optimization Technique** options: • **Optimization Technique Speed**—optimizes timing-critical portions of your design for performance at the cost of increasing area (logic and register utilization) • **Optimization Technique Balanced**—also optimizes the timing-critical portions of your design for performance, but the option allows only limited area increase • **Optimization Technique Area**—optimizes your design only for area

1.11. Fitter Settings Reference

Use Fitter settings to customize the place and route of your design. Click **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Fitter)** to access Fitter settings.

Table 33. Advanced Fitter Settings (1 of 8)

Option	Description
ALM Register Packing Effort	Guides aggressiveness of the Fitter in packing ALMs during register placement. Use this option to increase secondary register locations. Increasing ALM packing density may lower the number of ALMs needed to fit the design, but it may also reduce routing flexibility and timing performance. • **Low**—the Fitter avoids ALM packing configurations that combine LUTs and registers which have no direct connectivity. Avoiding these configurations may improve timing performance but increases the number of ALMs to implement the design. • **Medium**—the Fitter allows some configurations that combine unconnected LUTs and registers to be implemented in ALM locations. The Fitter makes more use of secondary register locations within the ALM. • **High**—the Fitter enables all legal and desired ALM packing configurations. In dense designs, the Fitter automatically increases the ALM register packing effort as required to enable the design to fit.
Advanced Physical Synthesis	Enables the Physical Synthesis engine that includes combinational and sequential optimization during fitting to improve circuit performance.
Allow Delay Chains	Allows the Fitter to choose the optimal delay chain to meet t_{SU} and t_{CO} timing requirements for all I/O elements. Enabling this option may reduce the number of t_{SU} violations, while introducing a minimal number of t_H violations. Enabling this option does not override delay chain settings on individual nodes.

continued...

Option	Description
Allow DSP Retiming	Allow retiming through DSP blocks.
Allow Early Global Retiming in the Fitter	Allows the Compiler to run global retiming early in the Fitter.
Allow Hyper-Aware Register Chain Area Optimizations in the Fitter	Reduces ALM usage by automatically forcing some back-to-back registers into Hyper Registers. Turning on this area reduction technique may reduce performance and increase compile time.
Allow RAM Retiming	Allow retiming through RAM blocks.
Allow Register Duplication	Allows the Compiler to duplicate registers to improve design performance. When you enable this option, the Compiler copies registers and moves some fan-out to this new node. This optimization improves routability and can reduce the total routing wire in nets with many fan-outs. If you disable this option, this disables optimizations that retime registers. *Note:* Not available for Intel Stratix 10 devices.
Allow Register Merging	Allows the Compiler to remove registers that are identical to other registers in the design. When you enable this option, in cases where two registers generate the same logic, the Compiler deletes one register, and the remaining registers fan-out to the deleted register's destinations. This option is useful if you want to prevent the Compiler from removing intentional use of duplicate registers. If you disable register merging, the Compiler disables optimizations that retime registers. *Note:* Not available for Intel Stratix 10 devices.
Auto Delay Chains for High Fanout Input Pins	Allows the Fitter to choose how to optimize the delay chains for high fan-out input pins. You must enable **Auto Delay Chains** to enable this option. Enabling this option may reduce the number of t_{SU} violations, but the compile time increases significantly, as the Fitter tries to optimize the settings for all fan-outs.
Auto Fit Effort Desired Slack Margin	Specifies the default worst-case slack margin the Fitter maintains for. If the design is likely to have at least this much slack on every path, the Fitter reduces optimization effort to reduce compilation time. *Note:* Not available for Intel Stratix 10 devices.

Table 34. Advanced Fitter Settings (2 of 8)

Option	Description
Auto Global Clock	Allows the Compiler to choose the global clock signal. The Compiler chooses the signal that feeds the most clock inputs to flip-flops. This signal is available throughout the device on the global routing paths. To prevent the Compiler from automatically selecting a particular signal as global clock, set the **Global Signal** option to **Off** on that signal.
Auto Global Register Control Signals	Allows the Compiler to choose global register control signals. The Compiler chooses signals that feed the most control signal inputs to flip-flops (excluding clock signals) as the global signals. These global signals are available throughout the device on the global routing paths. Depending on the target device family, these control signals can include asynchronous clear and load, synchronous clear and load, clock enable, and preset signals. If you want to prevent the Compiler from automatically selecting a particular signal as a global register control signal, set the **Global Signal** option to **Off** on that signal.
Auto Packed Registers	Allows the Compiler to combine a register and a combinational function, or to implement registers using I/O cells, RAM blocks, or DSP blocks instead of logic cells. This option controls how aggressively the Fitter combines registers with other function blocks to reduce the area of the design. Generally, the **Auto** or **Sparse Auto** settings are appropriate.

continued...

Option	Description
	The other settings limit the flexibility of the Fitter to combine registers with other function blocks and can result in no fits. • **Auto**—the Fitter attempts to achieve the best performance with good area. If necessary, the Fitter combines additional logic to reduce the area of the design to within the current device. • **Sparse Auto**—the Fitter attempts to achieve the highest performance, but may increase device usage without exceeding the device logic capacity. • **Off**—the Fitter does not combine registers with other functions. The **Off** setting severely increases the area of the design and may cause a no fit. • **Sparse**—the Fitter combines functions in a way which improves performance for many designs. • **Normal**—the Fitter combines functions that are expected to maximize design performance and reduce area. • **Minimize Area**—the Fitter aggressively combines unrelated functions to reduce the area required for placing the design, at the expense of performance. • **Minimize Area with Chains**—the Fitter even more aggressively combines functions that are part of register cascade chains or can be converted to register cascade chains. If this option is set to any value but **Off**, registers combine with I/O cells to improve I/O timing. This remains true provided that the **Optimize IOC Register Placement For Timing** option is enabled.
Auto RAM to MLAB Conversion	Specifies whether the Fitter converts RAMs of **Auto** block type to use LAB locations. If this option is set to **Off,** only MLAB cells or RAM cells with a block type setting of **MLAB** use LAB locations to implement memory.
Auto Register Duplication	Allows the Fitter to automatically duplicate registers within a LAB that contains empty logic cells. This option does not alter the functionality of the design. The Compiler ignores the **Auto Register Duplication** option if you select **OFF** as the setting for the **Logic Cell Insertion -- Logic Duplication** logic option. Turning on this option allows the **Logic Cell Insertion -- Logic Duplication** logic option to improve a design's routability, but can make formal verification of a design more difficult. *Note:* Not available for Intel Stratix 10 devices.

Table 35. Advanced Fitter Settings (3 of 8)

Option	Description
Enable Auto-Pipelining	Turns on the auto-pipelining and latency-insensitive false path feature. Use this setting in conjunction with the **Maximum Additional Pipelining** and optional **Additional Pipelining Group** assignments in the Assignment Editor to automatically add pipeline registers at the locations you specify. *Note:* Available only for Intel Stratix 10 devices.
Enable Bus-Hold Circuitry	Enables bus-hold circuitry during device operation. When this option is **On**, a pin retains its last logic level when it is not driven, and does not go to a high impedance logic level. Do not use this option at the same time as the **Weak Pull-Up Resistor** designs, enables location to the Critical Chain Viewer from the Fast option. The Compiler ignores this option if you apply it to anything other than a pin.
Enable Critical Chain Viewer	Enables critical chain visualization in the Fast Forward Timing Closure Recommendations report for Intel Stratix 10 devices.
Equivalent RAM and MLAB Paused Read Capabilities	Specifies whether RAMs implemented in MLAB cells must have equivalent paused read capabilities as RAMs implemented in block RAM. Pausing a read is the ability to keep around the last read value when reading is disabled. Allowing differences in paused read capabilities provides the Fitter more flexibility in implementing RAMs using MLAB cells. To allow the Fitter the most flexibility in deciding which RAMs are implemented using MLAB cells, set this option to **Don't Care**. The following options are available: • **Don't Care**—the Fitter can convert RAMs to MLAB cells, even if they do not have equivalent paused read capabilities to a block RAM implementation. The Fitter generates an information message about RAMs with different paused read capabilities. • **Care**—the Fitter does not convert RAMs to MLAB cells unless they have the equivalent paused read capabilities to a block RAM implementation.

continued...

Option	Description
Equivalent RAM and MLAB Power Up	Specifies whether RAMs implemented in MLAB cells must have equivalent power-up conditions as RAMs implemented in block RAM. Power-up conditions occur when the device powers-up or globally resets. Allowing non-equivalent power-up conditions provides the Fitter more flexibility in implementing RAMs using MLAB cells. To allow the Fitter the most flexibility in deciding which RAMs are implemented using MLAB cells, set this option to **Auto** or **Don't Care**. The following options are available: • **Auto**—the Fitter may convert RAMs to MLAB cells, even if the MLAB cells lack equivalent power-up conditions to a block RAM implementation. The Fitter also outputs a warning message about RAMs with non-equivalent power up conditions. • **Don't Care**—the same behavior as **Auto** applies, but the message is an information message. • **Care**—the Fitter does not convert RAMs to MLAB cells unless they have equivalent power up conditions to a block RAM implementation.
Final Placement Optimizations	Specifies whether the Fitter performs final placement optimizations. Performing final placement optimizations may improve timing and routability, but may also require longer compilation time.
Fitter Aggressive Routability Optimizations	Specifies whether the Fitter aggressively optimizes for routability. Performing aggressive routability optimizations may decrease design speed, but may also reduce routing wire usage and routing time. The **Automatically** setting allows the Fitter to decide whether aggressive routability is beneficial.

Table 36. Advanced Fitter Settings (4 of 8)

Option	Description
Fitter Effort	Specifies the level of physical synthesis optimization during fitting: • **Auto**—adjusts the Fitter optimization effort to minimize compilation time, while still achieving the design timing requirements. Use the **Auto Fit Effort Desired Slack Margin** option to apply sufficient optimization effort to achieve additional timing margin. • **Standard**—uses maximum effort regardless of the design's requirements, leading to higher compilation time and more margin on easier designs. For difficult designs, **Auto** and **Standard** both use maximum effort. *Note:* Not available for Intel Stratix 10 devices.
Fitter Initial Placement Seed	Specifies the seed for the current design. The value can be any non-negative integer value. By default, the Fitter uses a seed of 1. The Fitter uses the seed as the initial placement configuration when optimizing design placement to meet timing requirements f_{MAX}. Because each different seed value results in a somewhat different fit, you can try several different seeds to attempt to obtain superior fitting results. The seeds that lead to the best fits for a design may change if the design changes. Also, changing the seed may or may not result in a better fit. Therefore, specify a seed only if the Fitter is not meeting timing requirements by a small amount. *Note:* You can also use the Design Space Explorer II (DSEII) to sweep complex flow parameters, including the seed, in the Intel Quartus Prime software to optimize design performance.
Logic Cell Insertion	Allows the Fitter to automatically insert buffer logic cells between two nodes without altering the functionality of the design. The Compiler creates buffer logic cells from unused logic cells in the device. This option also allows the Fitter to duplicate a logic cell within a LAB when there are unused logic cells available in a LAB. Using this option can increase compilation time. The default setting of **Auto** allows these operations to run when the design requires them to fit the design. *Note:* Not available for Intel Stratix 10 devices.
MLAB Add Timing Constraints for Mixed-Port Feed-Through Mode Setting Don't Care	Specifies whether the Timing Analyzer evaluates timing constraints between the write and the read operations of the MLAB memory block. Performing a write and read operation simultaneously at the same address might result in metastability issues because no timing constraints between those operations exist by default. Turning on this option introduces timing constraints between the write and read operations on the MLAB memory block and

continued...

Option	Description
	thereby avoids metastability issues. However, turning on this option degrades the performance of the MLAB memory blocks. If your design does not perform write and read operations simultaneously at the same address, you do not need to set this option.
Number of Example Nodes Reported in Fitter Messages	Allows you to specify the maximum number of example nodes Fitter messages should display.

Table 37. Advanced Fitter Settings (5 of 8)

Option	Description
Optimize Design for Metastability	This setting improves the reliability of the design by increasing its Mean Time Between Failures (MTBF). When you enable this setting, the Fitter increases the output setup slacks of synchronizer registers in the design. This slack can exponentially increase the design MTBF. This option only applies when using the Timing Analyzer for timing-driven compilation. Use the Timing Analyzer `report_metastability` command to review the synchronizers detected in your design and to produce MTBF estimates.
Optimize Hold Timing	Directs the Fitter to optimize hold time within a device to meet timing requirements and assignments. The following settings are available: • **I/O Paths and Minimum TPD Paths—**directs the Fitter to meet the following timing requirements and assignments: — t_H from I/O pins to registers. — Minimum t_{CO} from registers to I/O pins. — Minimum t_{PD} from I/O pins or registers to I/O pins or registers. • **All Paths**—directs the Fitter to meet the following timing requirements and assignments: — t_H from I/O pins to registers. — Minimum t_{CO} from registers to I/O pins. — Minimum t_{PD} from I/O pins or registers to I/O pins or registers. When you disable the **Optimize Timing** logic option, the **Optimize Hold Timing** option is not available.
Optimize IOC Register Placement for Timing	Specifies whether the Fitter optimizes I/O pin timing by automatically packing registers into I/Os to minimize delays. • **Normal**—the Fitter opportunistically packs registers into I/Os that should improve I/O timing. • **Pack All I/O Registers**— the Fitter aggressively packs any registers connected to input, output, or output enable pins into I/Os, unless prevented by your constraints or other legality restrictions. • **Off**—performs no periphery to core optimization.
Optimize Multi-Corner Timing	Directs the Fitter to consider all timing corners during optimization to meet timing requirements. These timing delay corners include both fast-corner timing and slow-corner timing. By default, this option is **On**, and the Fitter optimizes designs considering multi-corner delays in addition to slow-corner delays. When this option is **Off**, the Fitter optimizes designs considering only slow-corner delays from the slow-corner timing model (slowest manufactured device for a given speed grade, operating in low-voltage conditions). Turning this option **On** typically creates a more robust design implementation across process, temperature, and voltage variations. When you turn **Off** the **Optimize Timing** option, the **Optimize Multi-Corner Timing** option is not available.
Optimize Timing	Specifies whether the Fitter optimizes to meet the maximum delay timing requirements (for example, clock cycle time). By default, this option is set to **Normal compilation**. Turning this option **Off** helps fit designs that with extremely high interconnect requirements. Turning this option **Off** can also reduce compilation time at the expense of timing performance (because the Fitter ignores the design's timing requirements). If this option is **Off**, other Fitter timing optimization options have no effect (such as **Optimize Hold Timing**).

Table 38. Advanced Fitter Settings (6 of 8)

Option	Description
Periphery to Core Placement and Routing Optimization	Specifies whether the Fitter should perform targeted placement and routing optimization on direct connections between periphery logic and registers in the FPGA core. The following options are available: • **Auto**—the Fitter automatically identifies transfers with tight timing windows, places the core registers, and routes all connections to or from the periphery. The Fitter performs these placement and routing decisions before the rest of core placement and routing. This sequence ensures that these timing-critical connections meet timing, and also avoids routing congestion. • **On**— the Fitter optimizes all transfers between the periphery and core registers, regardless of timing requirements. Do not set this option to **On** globally. Instead, use the Assignment Editor to assign optimization to a targeted set of nodes or entities. • **Off**—the Fitter performs no periphery to core optimization. *Note:* Not available for Intel Stratix 10 devices.
Physical Placement Effort	Controls how much effort the Fitter spends during advanced physical placement optimization. **High** and **Maximum** effort settings result in additional compile time to further optimization the placement solution.
Placement Effort Multiplier	Specifies the relative time the Fitter spends in placement. The default value is 1.0, and legal values must be greater than 0. Specifying a floating-point number allows you to control the placement effort. A higher value increases CPU time but may improve placement quality. For example, a value of '4' increases fitting time by approximately 2 to 4 times but may increase quality.
Power Optimization During Fitting	Directs the Fitter to perform optimizations targeted at reducing the total power devices consume. The available settings for power-optimized fitting are: • **Off**—performs no power optimizations. • **Normal compilation**—performs power optimizations that are unlikely to adversely affect compilation time or design performance. • **Extra effort**—performs additional power optimizations that might affect design performance or result in longer compilation time.

Table 39. Advanced Fitter Settings (7 of 8)

Option	Description
Programmable Power Maximum High-Speed Fraction of Used LAB Tiles	Sets the upper limit on the fraction of the high-speed LAB tiles. Legal values must be between 0.0 and 1.0. The default value is 1.0. A value of 1.0 means that there is no restriction on the number of high-speed tiles, and the Fitter uses the minimum number needed to meet the timing requirements of your design. Specifying a value lower than 1.0 might degrade timing quality, because some timing critical resources might be forced into low-power mode.
Programmable Power Technology Optimization	Controls how the Fitter configures tiles to operate in high-speed mode or low-power mode. The following options are available: • **Automatic**—specifies that the Fitter minimizes power without sacrificing timing performance. • **Minimize Power Only**—specifies that the Fitter sets the maximum number of tiles to operate in low-power mode. • **Force All Used Tiles to High Speed**—specifies that the Fitter sets all used tiles to operate in high-speed mode. • **Force All Tiles with Failing Timing Paths to High Speed**—sets all failing paths to high-speed mode. For designs that meet timing, the behavior of this setting is similar to the **Automatic** setting. For designs that fail timing, all paths with negative slack are put in high-speed mode. This mode likely does not increase the speed of the design, and it may increase static power consumption. This mode may assist in determining which logic paths need to be re-designed to close timing.

continued...

Option	Description
	Note: Not available for Intel Stratix 10 devices.
Router Timing Optimization Level	Controls how aggressively the router tries to meet timing requirements. Setting this option to **Maximum** can increase design speed slightly, at the cost of increased compile time. Setting this option to **Minimum** can reduce compile time, at the cost of slightly reduced design speed. The default value is **Normal**.
Run Early Place during compilation	Enables the Early Place Fitter stage during full compilation. Turning on this setting may increase Fitter processing time.

Table 40. Advanced Fitter Settings (8 of 8)

Option	Description
Synchronizer Identification	Specifies how the Compiler identifies synchronization register chain registers for metastability analysis. A synchronization register chain is a sequence of registers with the same clock with no fan-out in between, which is driven by a pin or logic from another clock domain. The following options are available: • **Off**—the Timing Analyzer does not identify the specified registers, or the registers within the specified entity, as synchronization registers. • **Auto**—the Timing Analyzer identifies valid synchronization registers that are part of a chain with more than one register that contains no combinational logic. Use the **Auto** setting to generate a report of possible synchronization chains in your design. • **Forced if Asynchronous**—the Timing Analyzer identifies synchronization register chains if the software detects an asynchronous signal transfer, even if there is combinational logic or only one register in the chain. • **Forced**—the Timing Analyzer identifies the specified register, or all registers within the specified entity, as synchronizers. Only apply the **Forced** option to the entire design. Otherwise, all registers in the design identify as synchronizers. The Fitter optimizes the registers that it identifies as synchronizers for improved Mean Time Between Failure (MTBF), provided that you enable **Optimize Design for Metastability**. If a synchronization register chain is identified with the **Forced** or **Forced if Asynchronous** option, then the Timing Analyzer reports the metastability MTBF for the chain when it meets the design timing requirements.
Treat Bidirectional Pin as Output Pin	Specifies that the Fitter treats the bidirectional pin as an output pin, meaning that the input path feeds back from the output path.
Use Checkered Pattern as uninitialized RAM Content	Loads a checkered pattern as initial RAM content into all RAM blocks without specified RAM content that supports content initialization. Turning on this option does not affect simulation, which may cause on-chip behavior to differ from simulation results.
Weak Pull-Up Resistor	Enables the weak pull-up resistor when the device is operating in user mode. This option pulls a high-impedance bus signal to VCC. Do not enable this option simultaneously with the **Enable Bus-Hold Circuitry** option. The Fitter ignores this option if you apply to anything other than a pin.

1.12. Design Compilation Revision History

This document has the following revision history.

Document Version	Intel Quartus Prime Version	Changes
2019.07.02	19.1	• Made minor changes in "Fractal Synthesis Optimization" topic. • Added a note in step 3a of "Running Synthesis" about enabling fractal synthesis project-wide. • Added details about synthesis of `PRESERVE_FANOUT_FREE_NODE` to "Partial Reconfiguration Design Guidelines." • Corrected typo in "Step 3: Run Fast Forward Compile and Hyper-Retiming."
2019.04.01	19.1	• In "Running Synthesis", removed a step about enabling fractal synthesis project-wide. • Updated the "Fractal Synthesis Optimization" topic to describe signed multiplication feature that is now supported by multiplier regularization and arithmetic packing algorithms.
2019.01.03	18.1.0	• Added snapshot description to "Compilation Overview"and linked to content from "Exporting a Design Partition" and "Exporting a Version-Compatible Compilation Database."
2018.10.19	18.1.0	• Described dependency of Rapid Recompile on Enable Intermediate Fitter Snapshots option.
2018.09.24	18.1.0	• Described option to enable or disable intermediate Fitter snapshots and updated descriptions of compilation flows and dashboard accordingly. • Added "Exporting Compilation Results section and subtopics." • Described migration of full chip database in "Exporting a Version-Compatible Compilation Database" topic. • Described automated `.qdb` partition export in "Exporting a Design Partition" topic. • Described viewing QDB file metadata in "Viewing Quartus Database File Information." • Added "Fractal Synthesis Optimization" topic and updated "Running Synthesis" topic steps for new option. • Described new Compiler Optimization Modes and described notice that appears for extended optimization modes added via `.qsf`. • Described new Global Signal Visualization Report. • Added "Factors Affecting Compilation Results" topic. • Added "Using the Compilation Dashboard" topic. • Added description of Enable Auto-Pipelining setting. • Added description of Enable Formal Verification Support to "Advanced Synthesis Settings." • Added description of Report PR Initial Values as Errors option to "Advanced Synthesis Settings." • Added description of Size of the Latch Report option to "Advanced Synthesis Settings." • Added description of Size of the PR Initial Conditions Report option to "Advanced Synthesis Settings." • Added description of Advanced Physical Synthesis option to "Fitter Settings Reference." • Added description of Allow DSP Retiming option to "Fitter Settings Reference." • Added description of Allow Early Global Retiming in the Fitter option to "Fitter Settings Reference." • Added description of Allow Hyper-Aware Register Chain Area Optimizations in the Fitter option to "Fitter Settings Reference." • Added description of Allow RAM Retiming option to "Fitter Settings Reference." • Added description of Number of Example Nodes Reported in Fitter Messages option to "Fitter Settings Reference." • Added description of Physical Placement Effort option to "Fitter Settings Reference."

continued...

Document Version	Intel Quartus Prime Version	Changes
		• Added description of Use Checkered Pattern as uninitialized RAM Content option to "Fitter Settings Reference." • Updated description of Safe State Machine option for Auto setting. • Removed support for Ignore ROW GLOBAL Buffers option. • Removed support for Ignore CARRY Buffers option. • Removed support for Ignore CASCADE Buffers option.
2018.05.07	18.0.0	• Updated Optimization Modes topic to add Compile Time (Aggressive). • Relocated concurrent analysis content from the *Early Place Flow* topic to a new *Concurrent Analysis During Synthesis or Fitting* topic. • Rapid Recompile now supports Intel Stratix 10 devices. • Enhanced description of Retime Stage Reports. • Enhanced description of Retime Stage to include classic register retiming.

Table 41. Document Revision History

Date	Version	Changes
2017.11.06	17.1.0	• Added support for Intel Stratix 10 Hyper-Aware design flow, Hyper-Retiming, Fast Forward compilation, and Fast Forward Viewer. • Added Advanced HyperFlex Settings topic. • Added Retiming Restrictions and Workarounds topic. • Added statement about Fast Forward compilation support for retiming across RAM and DSP blocks. • Added Concurrent Analysis topic. • Added Analyzing Fitter Snapshots topic. • Added Compilation Dashboard Early Place stage control image. • Added Running late_place After Early Place topic. • Updated for latest Intel naming conventions.
2017.05.08	17.0.0	• Added reference to initial compilation support for Cyclone 10 GX devices. • Described concurrent analysis following Early Place. • Updated Compilation Dashboard images for Timing Analyzer, Report, Setting, and Concurrent Analysis controls. • Updated description for Auto DSP Block Replacement in Advanced Synthesis Settings. • Updated Advanced Fitter Settings for Allow Register Retiming, and for removal of obsolete SSN Optimization option. • Added Prevent Register Retiming topic. • Added Preserve Registers During Synthesis topic. • Removed limitation for **Safe State Machine** logic option. • Added references to Partial Reconfiguration and Block-Based Design Flows.
2016.10.31	16.1.0	• Implemented Intel re-branding. • Described Compiler snapshots and added Analyzing Snapshot Timing topic. • Updated project directory structure diagram. • Described new Fitter stage menu commands and reports. • Added description of Early Place Flow, Implement Flow, and Finalize Flow.

continued...

Date	Version	Changes
		• Added description of Incremental Optimization in the Fitter. • Reorganized order of topics in chapter. • Removed deprecated **Per-Stage Compilation (Beta)** Compilation Flow.
2016.05.03	16.0.0	• Added description of Fitter Plan, Place and Route stages, reporting, and optimization. • Added **Per-Stage Compilation (Beta)** Compilation Flow • Added Compilation Dashboard information. • Removed support for **Safe State Machine** logic option. Encode safe states in RTL. • Added Generating Dynamic Synthesis Reports topic. • Updated Quartus project directory structure.
2015.11.02	15.1.0	• First version of document.

Related Information

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

2. Reducing Compilation Time

You can employ various techniques to reduce to time required for synthesis and fitting in the Intel Quartus Prime Compiler.

2.1. Compilation Time Advisor

A Compilation Time Advisor is available in the Intel Quartus Prime GUI by clicking **Tools ➤ Advisors ➤ Compilation Time Advisor**. This chapter describes all the compilation time optimizing techniques available in the Compilation Time Advisor.

2.2. Strategies to Reduce the Overall Compilation Time

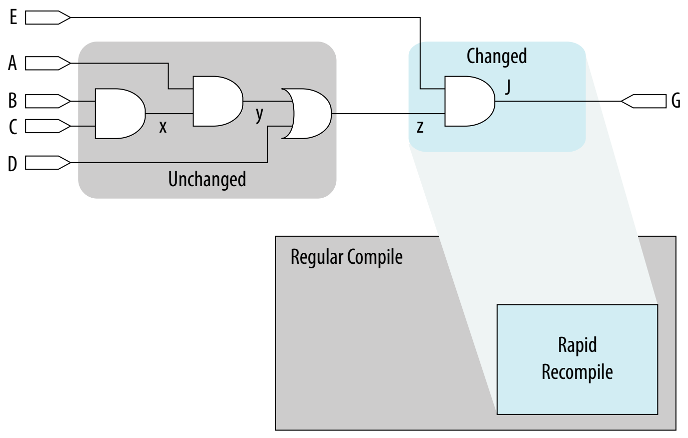
You can use the following strategies to reduce the overall time required to compile your design:

- Parallel compilation (for systems with multiple processor cores)
- Rapid Recompile and Smart Compilation reuse results from a previous compilation to reduce overall compilation time

2.2.1. Running Rapid Recompile

During Rapid Recompile the Compiler reuses previous synthesis and fitting results whenever possible, and does not reprocess unchanged design blocks. Use Rapid Recompile to reduce timing variations and the total recompilation time after making small design changes.

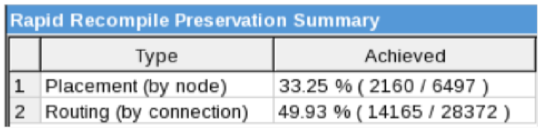
Figure 65. Rapid Recompile



To run Rapid Recompile, follow these steps:

1. Prior to initial compilation, click **Assignments ➤ Settings ➤ Compiler Settings** and turn on **Enable Intermediate Fitter Snapshots**. This option must be enabled to subsequently use the Rapid Recompile feature.
2. To start Rapid Recompile following an initial compilation (or after running the Route stage of the Fitter), click **Processing ➤ Start ➤ Start Rapid Recompile**. Rapid Recompile implements the following types of design changes without full recompilation:
   - Changes to nodes tapped by the Signal Tap Logic Analyzer
   - Changes to combinational logic functions
   - Changes to state machine logic (for example, new states, state transition changes)
   - Changes to signal or bus latency or addition of pipeline registers
   - Changes to coefficients of an adder or multiplier
   - Changes register packing behavior of DSP, RAM, or I/O
   - Removal of unnecessary logic
   - Changes to synthesis directives
3. Click the Rapid Recompile Preservation Summary report to view detailed information about the percentage of preserved compilation results.

Figure 66. Rapid Recompile Preservation Summary



Rapid Recompile Preservation Summary

	Type	Achieved
1	Placement (by node)	33.25 % (2160 / 6497)
2	Routing (by connection)	49.93 % (14165 / 28372)

2.2.2. Enabling Multi-Processor Compilation

The Compiler can detect and use multiple processors to reduce total compilation time. You specify the number of processors the Compiler uses. The Intel Quartus Prime software can use up to 16 processors to run algorithms in parallel. The Compiler uses parallel compilation by default. To reserve some processors for other tasks, specify a maximum number of processors that the software uses.

This technique reduces the compilation time by up to 10% on systems with two processing cores, and by up to 20% on systems with four cores. When running timing analysis independently, two processors reduce the timing analysis time by an average of 10%. This reduction reaches an average of 15% when using four processors.

The Intel Quartus Prime software does not necessarily use all the processors that you specify during a given compilation. Additionally, the software never uses more than the specified number of processors. This fact enables you to work on other tasks without slowing down your computer. The use of multiple processors does not affect the quality of the fit. For a given Fitter seed, and given **Maximum processors allowed** setting on a specific design, the fit is exactly the same and deterministic. This remains true, regardless of the target machine, and the number of available processors. Different **Maximum processors allowed** specifications produce different results of the same quality. The impact is similar to changing the Fitter seed setting.

To enable multiprocessor compilation, follow these steps:

1. Open or create an Intel Quartus Prime project.
2. Click **Assignments ➤ Settings ➤ Compilation Process Settings**.
3. Under **Parallel compilation**, specify options for the number of processors the Compiler uses.
4. View detailed information about processor use in the Parallel Compilation report following compilation.

   To specify the number of processors for compilation at the command line, use the following Tcl command in your script:

   ```
   set_global_assignment -name NUM_PARALLEL_PROCESSORS <value>
   ```

   In this case, *<value>* is an integer from 1 to 16.

   If you want the Intel Quartus Prime software to detect the number of processors and use all the processors for the compilation, include the following Tcl command in your script:

   ```
   set_global_assignment -name NUM_PARALLEL_PROCESSORS ALL
   ```

   Note: The Compiler detects Intel Hyper-Threading® Technology (Intel® HT Technology) as a single processor. If your system includes a single processor with Intel HT Technology, set the number of processors to one. Do not use the Intel HT Technology for Intel Quartus Prime compilations.

2.2.3. Using Block-Based Compilation

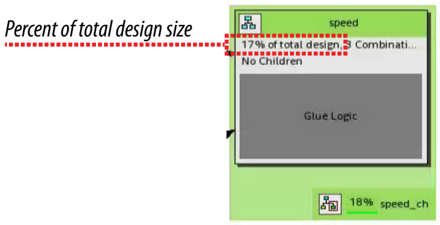
During the design process, you can isolate functional blocks that meet placement and timing requirements from others still undergoing change and optimization. By isolating functional blocks into partitions, you can apply optimization techniques to selected areas only compile those areas.

To create partitions dividing functional blocks:

1. In the Design Partition Planner, identify blocks of a size suitable for partitioning.

   In general, a partition represents roughly 15 to 20 percent of the total design size. Use the information area below the bar at the top of each entity.

Figure 67. Entity representation in the Design Partition Planner


Percent of total design size
speed
17% of total design, 3 Combinati...
No Children
Glue Logic
18%
speed_ch

2. Extract and collapse entities as necessary to achieve stand-alone blocks
3. For each entity of the desired size containing related blocks of logic, right-click the entity and click **Create Design Partition** to place that entity in its own partition.

   The goal is to achieve partitions containing related blocks of logic.

Related Information

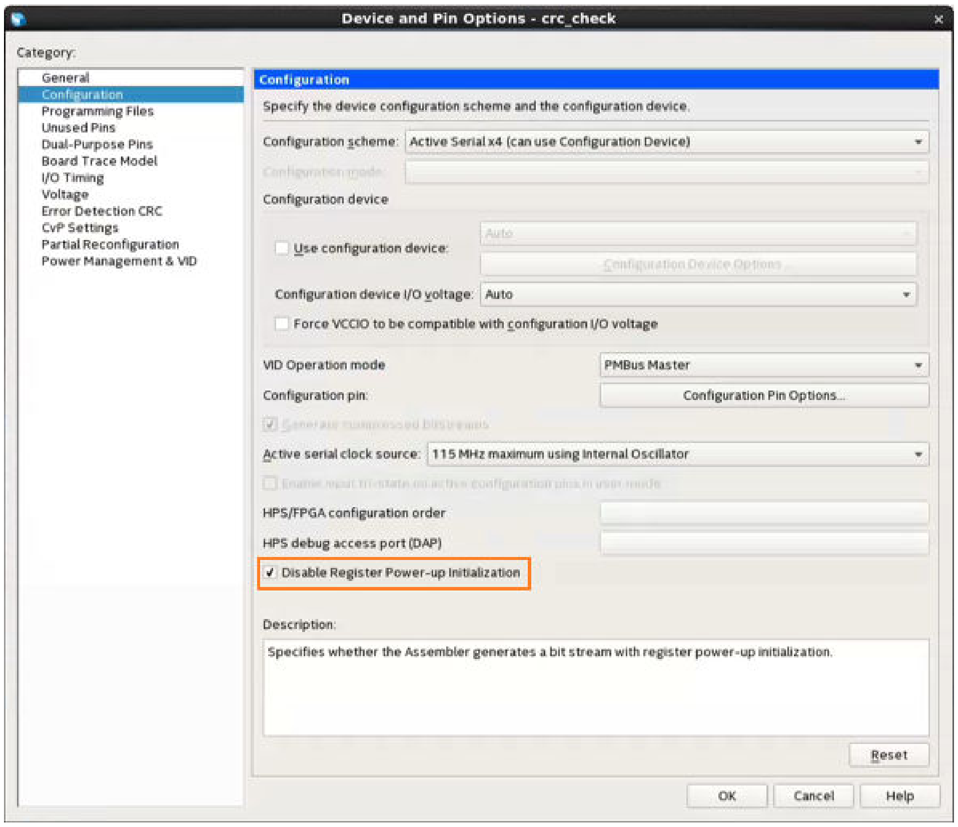
Intel Quartus Prime Pro Edition User Guide: Block-Based Design

2.2.4. Disabling the Register Power-up Initialization

By disabling the register power-up initialization, you can speed up the bitstream initialization process, reduce the bitstream size (allowing you to select smaller configuration device), and reduce the configuration time.

Important:
- This is available only for Intel Stratix 10 devices.
- Optimizations that rely on power-up states are disabled.
- Bitstream assembler creates bitstreams without the register power-up initialization.

To disable the initialization, enable the **Disable Register Power-up Initialization** option through the **Device and Pin Options** dialog.



Device and Pin Options - crc_check

Category: General, Configuration, Programming Files, Unused Pins, Dual-Purpose Pins, Board Trace Model, I/O Timing, Voltage, Error Detection CRC, CvP Settings, Partial Reconfiguration, Power Management & VID

Configuration

Specify the device configuration scheme and the configuration device.

Configuration scheme: Active Serial x4 (can use Configuration Device)

Configuration mode:

Configuration device

Use configuration device: Auto

Configuration device I/O voltage: Auto

Force VCCIO to be compatible with configuration I/O voltage

VID Operation mode: PMBus Master

Configuration pin: Configuration Pin Options...

[✓] Generate compressed bitstreams

Active serial clock source: 115 MHz maximum using Internal Oscillator

Enable input tri-state on active configuration pins in user mode

HPS/FPGA configuration order

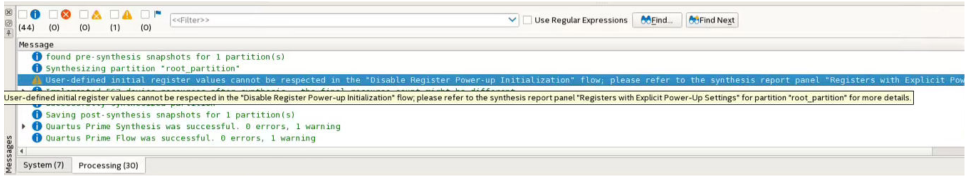
HPS debug access port (DAP)

[✓] Disable Register Power-up Initialization

Description:

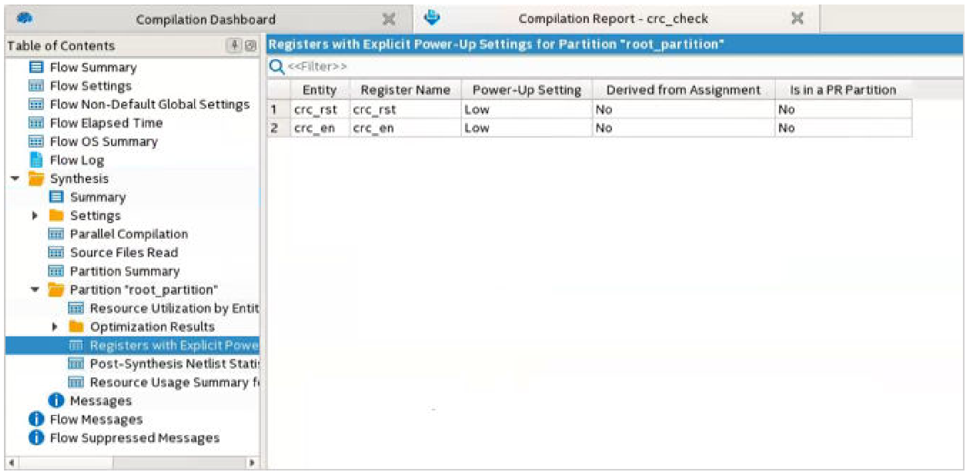
Specifies whether the Assembler generates a bit stream with register power-up initialization.

When you enable the **Disable Register Power-up Initialization** option, Synthesis prints a warning for registers with power-up care, as shown in the following image:



```
Message
found pre-synthesis snapshots for 1 partition(s)
Synthesizing partition "root_partition"
User-defined initial register values cannot be respected in the "Disable Register Power-up Initialization" flow; please refer to the synthesis report panel "Registers with Explicit Pow
User-defined initial register values cannot be respected in the "Disable Register Power-up Initialization" flow; please refer to the synthesis report panel "Registers with Explicit Power-Up Settings" for partition "root_partition" for more details.
Saving post-synthesis snapshots for 1 partition(s)
Quartus Prime Synthesis was successful. 0 errors, 1 warning
Quartus Prime Flow was successful. 0 errors, 1 warning
```

To view registers with defined power-up states that cannot be preserved, refer to the **Registers with Explicit Power-up Settings** dialog, as shown in the following image:



Registers with Explicit Power-Up Settings for Partition "root_partition"

	Entity	Register Name	Power-Up Setting	Derived from Assignment	Is in a PR Partition
1	crc_rst	crc_rst	Low	No	No
2	crc_en	crc_en	Low	No	No

For more information about the reset strategy required when you no longer can rely on registers' power-up states to reset the designs, refer to Partial Reconfiguration Design Flow in *Intel Quartus Prime Pro Edition User Guide: Partial Reconfiguration*.

Related Information

- Intel Stratix 10 Configuration Bit Stream Sizes
- Intel Stratix 10 Reset Release IP

2.3. Reducing Synthesis Time and Synthesis Netlist Optimization Time

You can reduce synthesis time without affecting the Fitter time by reducing your use of netlist optimizations. For tips on reducing synthesis time when using third-party EDA synthesis tools, refer to your synthesis software’s documentation.

2.3.1. Settings to Reduce Synthesis Time and Synthesis Netlist Optimization Time

Synthesis netlist and physical synthesis optimization settings can significantly increase the overall compilation time for large designs. Refer to Analysis and Synthesis messages to determine the length of optimization time.

If your design already meets performance requirements without synthesis netlist or physical synthesis optimizations, turn off these options to reduce compilation time. If you require synthesis netlist optimizations to meet performance, optimize partitions of your design hierarchy separately to reduce the overall time spent in Analysis and Synthesis.

2.3.2. Use Appropriate Coding Style to Reduce Synthesis Time

Your HDL coding style can also affect the synthesis time. For example, if you want to infer RAM blocks from your code, you must follow the guidelines for inferring RAMs. If RAM blocks are not inferred properly, the software implements those blocks as registers.

If you are trying to infer a large memory block, the software consumes more resources in the FPGA. This can cause routing congestion and increasing compilation time significantly. If you see high routing utilizations in certain blocks, it is a good idea to review the code for such blocks.

2.4. Reducing Placement Time

The time required to place a design depends on two factors: the number of ways the logic in your design can be placed in the device, and the settings that control the amount of effort required to find a good placement.

You can reduce the placement time by changing the settings for the placement algorithm.

Sometimes there is a trade-off between placement time and routing time. Routing time can increase if the placer does not run long enough to find a good placement. When you reduce placement time, ensure that it does not increase routing time and negate the overall time reduction.

2.4.1. Placement Effort Multiplier Settings

You can control the amount of time the Fitter spends in placement by reducing with the **Placement Effort Multiplier** option.

Click **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Fitter)** and specify a value for Placement Effort Multiplier. The default is `1.0`. Legal values must be greater than 0 and can be non-integer values. Numbers between 0 and 1 can reduce fitting time, but also can reduce placement quality and design performance.

2.5. Reducing Routing Time

The routing time is usually not a significant amount of the compilation time. The time required to route a design depends on three factors: the device architecture, the placement of your design in the device, and the connectivity between different parts of your design.

If your design requires a long time to route, perform one or more of the following actions:

- Check for routing congestion.
- Turn off **Fitter Aggressive Routability Optimization**.

2.5.1. Identifying Routing Congestion with the Chip Planner

To identify areas of routing congestion in your design:

1. Click **Tools ➤ Chip Planner**.
2. To view the routing congestion in the Chip Planner, double-click the **Report Routing Utilization** command in the **Tasks** list.
3. Click **Preview** in the **Report Routing Utilization** dialog box to preview the default congestion display.
4. Change the **Routing utilization type** to display congestion for specific resources. The default display uses dark blue for 0% congestion and red for 100%.
5. Adjust the slider for **Threshold percentage** to change the congestion threshold level.

The Intel Quartus Prime compilation messages contain information about average and peak interconnect usage. Peak interconnect usage over 75%, or average interconnect usage over 60% indicate possible difficulties fitting your design. Similarly, peak interconnect usage over 90%, or average interconnect usage over 75%, indicate a high chance of not getting a valid fit.

2.5.1.1. Areas with Routing Congestion

Even if average congestion is not high, the design may have areas where congestion is high in a specific type of routing. You can use the Chip Planner to identify areas of high congestion for specific interconnect types.

- You can change the connections in your design to reduce routing congestion
- If the area with routing congestion is in a Logic Lock (Standard) region or between Logic Lock (Standard) regions, change or remove the Logic Lock (Standard) regions and recompile your design.
  - If the routing time remains the same, the time is a characteristic of your design and the placement
  - If the routing time decreases, consider changing the size, location, or contents of Logic Lock (Standard) regions to reduce congestion and decrease routing time.

Related Information

Intel Quartus Prime Pro Edition User Guide: Design Optimization

2.5.1.2. Congestion due to HDL Coding style

Sometimes, routing congestion may be a result of the HDL coding style used in your design. After identifying congested areas using the Chip Planner, review the HDL code for the blocks placed in those areas to determine whether you can reduce interconnect usage by code changes.

2.6. Reducing Static Timing Analysis Time

If you are performing timing-driven synthesis, the Intel Quartus Prime software runs the Timing Analyzer during Analysis and Synthesis.

The Intel Quartus Prime Fitter also runs the Timing Analyzer during placement and routing. If there are incorrect constraints in the Synopsys Design Constraints File (`.sdc`), the Intel Quartus Prime software may spend unnecessary time processing constraints several times.

- If you do not specify false paths and multicycle paths in your design, the Timing Analyzer may analyze paths that are not relevant to your design.
- If you redefine constraints in the `.sdc` files, the Timing Analyzer may spend additional time processing them. To avoid this situation, look for indications that Synopsis design constraints are being redefined in the compilation messages, and update the `.sdc` file.
- Ensure that you provide the correct timing constraints to your design, because the software cannot assume design intent, such as which paths to consider as false paths or multicycle paths. When you specify these assignments correctly, the Timing Analyzer skips analysis for those paths, and the Fitter does not spend additional time optimizing those paths.

2.7. Setting Process Priority

It might be necessary to reduce the computing resources allocated to the compilation at the expense of increased compilation time. It can be convenient to reduce the resource allocation to the compilation with single processor machines if you must run other tasks at the same time.

Related Information

Processing Page (Options Dialog Box)
In Intel Quartus Prime Help.

2.8. Reducing Compilation Time Revision History

Document Version	Intel Quartus Prime Version	Changes
2019.07.02	19.1	Added the *Using the No-Register Initialization Flow* topic.
2018.10.19	18.1.0	• Described dependency of Rapid Recompile on Enable Intermediate Fitter Snapshots option.
2017.11.06	17.1.0	• Added topic: Using Block-Based Compilation.

Date	Version	Changes
2017.05.08	17.0.0	• Clarified impact of multiprocessor compilation on fit quality. • Removed reference to deprecated Fitter Effort Logic Option. • Removed section: *Preserving Routing with Incremental Compilation*.
2016.10.31	16.1.0	• Implemented Intel rebranding.
2016.05.02	16.0.0	• Corrected typo in Using Parallel Compilation with Multiple Processors. • Removed information about deprecated physical synthesis options.
2015.11.02	15.1.0	Changed instances of *Quartus II* to *Intel Quartus Prime*.
2014.12.15	14.1.0	• Updated location of Fitter Settings, Analysis & Synthesis Settings, and Physical Synthesis Optimizations to Compiler Settings. • Added information about Rapid Recompile feature.
2014.08.18	14.0a10.0	Added restriction about smart compilation in Arria 10 devices.
June 2014	14.0.0	Updated format.
May 2013	13.0.0	Removed the "Limit to One Fitting Attempt", "Using Early Timing Estimation", "Final Placement Optimizations", and "Using Rapid Recompile" sections.

continued...

Send Feedback

Date	Version	Changes
		Updated "Placement Effort Multiplier Settings" section. Updated "Identifying Routing Congestion in the Chip Planner" section. General editorial changes throughout the chapter.
June 2012	12.0.0	Removed survey link.
November 2011	11.0.1	Template update.
May 2011	11.0.0	• Updated "Using Parallel Compilation with Multiple Processors". • Updated "Identifying Routing Congestion in the Chip Planner". • General editorial changes throughout the chapter.
December 2010	10.1.0	• Template update. • Added details about peak and average interconnect usage. • Added new section "Reducing Static Timing Analysis Time". • Minor changes throughout chapter.
July 2010	10.0.0	Initial release.

Related Information

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

3. Intel Quartus Prime Pro Edition User Guide Design Compilation Archives

If the table does not list a software version, the user guide for the previous software version applies.

Intel Quartus Prime Version	User Guide
19.1	Intel Quartus Prime Pro Edition User Guide Design Compilation
18.1	Intel Quartus Prime Pro Edition User Guide Design Compilation
18.0	Compiler User Guide Intel Quartus Prime Pro Edition

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, optimizing device resource usage, device floorplanning, and implementing engineering change orders (ECOs).
- 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.