2. Reducing Compilation Time...........................................................................................87

2.1. Factors Affecting Compilation Results.........................................................................87
2.2. Compilation Time Advisor..........................................................................................87
2.3. Strategies to Reduce the Overall Compilation Time..................................................87
    2.3.1. Running the ECO Compilation Flow.................................................................87
    2.3.2. Running Rapid Recompile..................................................................................88
    2.3.3. Enabling Multi-Processor Compilation.............................................................89
    2.3.4. Using Block-Based Compilation......................................................................89
    2.3.5. Disabling the Register Power-up Initialization....................................................90
2.4. Reducing Synthesis Time and Synthesis Netlist Optimization Time..............................92
    2.4.1. Settings to Reduce Synthesis Time and Synthesis Netlist Optimization Time....92
    2.4.2. Use Appropriate Coding Style to Reduce Synthesis Time................................92
2.5. Reducing Placement Time..........................................................................................93
    2.5.1. Placement Effort Multiplier Settings..................................................................93
2.6. Reducing Routing Time............................................................................................93
    2.6.1. Identifying Routing Congestion with the Chip Planner.......................................93
2.7. Reducing Static Timing Analysis Time........................................................................94
2.8. Setting Process Priority............................................................................................95
2.9. Reducing Compilation Time Revision History............................................................95


A. Intel Quartus Prime Pro Edition User Guides..................................................................98
1. Design Compilation

The Intel® Quartus® Prime Compiler synthesizes, places, and routes your design before generating device programming files. 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

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

- Incremental Fitter optimization—analyze and optimize after each Fitter stage to maximize performance and shorten total compilation time.
- Partial Reconfiguration—dynamic reconfiguration of a portion of the FPGA, while the remaining FPGA continues to function.
- Block-Based Design Flows—preservation and reuse of design blocks.
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

<table>
<thead>
<tr>
<th>Compilation Process</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP Generation</td>
<td>Identifies the status and version of IP components in the project. Reports outdated IP that require upgrade.</td>
</tr>
<tr>
<td>Analysis &amp; Synthesis</td>
<td>Synthesizes, optimizes, minimizes, and maps design logic to device resources. The &quot;synthesized&quot; snapshot preserves the results of this stage. Analysis &amp; Elaboration is a stage of Analysis &amp; Synthesis. This stage checks for design file and project errors.</td>
</tr>
<tr>
<td>Fitter (Place &amp; Route)</td>
<td>Assigns the placement and routing of the design to specific device resources, while honoring timing and placement constraints. The Fitter includes the following stages:</td>
</tr>
<tr>
<td></td>
<td>• Plan—places all periphery elements (such as I/Os and PLLs) and determines a legal clock plan, without core placement or routing. The &quot;planned&quot; snapshot preserves the stage results.</td>
</tr>
<tr>
<td></td>
<td>• Place—places all core elements in a legal location. The &quot;placed&quot; snapshot preserves the stage results.</td>
</tr>
<tr>
<td></td>
<td>• Route—creates all routing between the elements in the design. The &quot;routed&quot; snapshot preserves the stage results.</td>
</tr>
<tr>
<td></td>
<td>• Retime—moves (retimes) existing registers into Hyper-Registers for fine-grained performance improvement. The &quot;retimed&quot; snapshot preserves the stage results.</td>
</tr>
<tr>
<td></td>
<td>• 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 and Intel Agilex devices, performs post-Route fix-up. The &quot;final&quot; snapshot preserves the stage results.</td>
</tr>
<tr>
<td>Fast Forward Timing Closure</td>
<td>Generates detailed reports that estimate performance gains achievable by making specific RTL modifications.</td>
</tr>
<tr>
<td>Recommendations</td>
<td>Analyzes and validates the timing performance of all design logic with the Timing Analyzer.</td>
</tr>
<tr>
<td>Timing Analysis</td>
<td>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.</td>
</tr>
<tr>
<td>Power Analysis</td>
<td>Converts the Fitter’s placement and routing assignments into a programming image for the FPGA device.</td>
</tr>
<tr>
<td>Assembler</td>
<td>Generates output files for use in other EDA tools, as Integrating Other EDA Tools on page 54 describes.</td>
</tr>
</tbody>
</table>

*Note:* Each successive release of the Intel Quartus Prime software typically includes:
- Added support for new features in supported FPGA devices.
- Added support for new devices.
- Efficiency and performance improvements.
- Improvements to compilation time and resource use of the design software.

Due to these improvements, different versions of the Intel Quartus Prime Pro Edition, Intel Quartus Prime Standard Edition, and Intel Quartus Prime Lite Edition software can produce different programming files from release to release.

---

(1) Retiming and Fast Forward compilation available only for Intel Stratix 10 and Intel Agilex devices.
1.1.1. 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>
<thead>
<tr>
<th>Compiler Flow</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECO Compilation Flow</td>
<td>The Intel Quartus Prime Pro Edition software supports last-minute, targeted design changes (also known as engineering change orders (ECOs)), even after you fully compile the design. ECOs typically occur during the design verification stage. Refer to the Intel Quartus Prime Pro Edition User Guide: Design Optimization.</td>
</tr>
<tr>
<td>Incremental Optimization Flow</td>
<td>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.</td>
</tr>
<tr>
<td>Hyper-Aware Design Flow</td>
<td>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 and Intel Agilex devices.</td>
</tr>
<tr>
<td>Full Compilation Flow</td>
<td>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.</td>
</tr>
<tr>
<td>Partial Reconfiguration</td>
<td>Reconfigures a portion of the FPGA dynamically, while the remaining FPGA design continues to function.</td>
</tr>
<tr>
<td>Block-Based Design Flows</td>
<td>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.</td>
</tr>
</tbody>
</table>

**Related Information**
- Incremental Optimization Flow on page 19
- Intel Quartus Prime Pro Edition User Guide: Block-Based Design
- Full Compilation Flow on page 44
- Running the Fitter on page 11
- Fast Forward Compilation Flow on page 30

1.1.2. 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 2. Hierarchical Project Structure (Intel Stratix 10 Design)**

- `<My_Project>` - top-level project directory
- `qdb` - Intel Quartus project database
- `qdb_compiler` - Intel Quartus project 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
- 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

**Related Information**
- Exporting Compilation Results on page 45
- Creating a Design Partition on page 47

### 1.2. Design Synthesis

Design synthesis is the process that translates design source files into an atom netlist for mapping to device resources. You can specify various settings that affect synthesis processing. The Intel Quartus Prime Compiler's Analysis and Synthesis module synthesizes standards-compliant Verilog HDL (.v), VHDL (.vhd), and SystemVerilog (.sv). The Compiler can also synthesize 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
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.

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.

**Figure 3. Design Synthesis**

![Design Synthesis Diagram](image)

### 1.2.1. 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, perform the following steps.

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 the optimization goal, according to **Optimization Modes** on page 62.
• On the Compiler Settings page enable or disable the Enable Intermediate Fitter Snapshots option to preserve snapshots for the Plan, Place, Route, and Retime stages any time you run full compilation. The Compiler does not generate intermediate snapshots by default.

• To specify advanced synthesis settings, click Assignments ➤ Settings ➤ Compiler Settings, and then click Advanced Settings (Synthesis).

• Consider enabling fractal synthesis for arithmetic-intensive designs that exhaust all DSP resources, according to the guidelines in Fractal Synthesis Optimization on page 65.

3. To run synthesis, click Synthesis on the Compilation Dashboard.

Related Information
Synthesis Settings Reference on page 71

1.2.1.1. Preserving 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

<table>
<thead>
<tr>
<th>Assignment</th>
<th>Description</th>
<th>Allows Fitter Optimization?</th>
<th>Assignment Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preserve Registers in Synthesis</td>
<td>Prevents removal of registers during synthesis. This setting does not affect retiming or other optimizations in the Fitter.</td>
<td>Yes</td>
<td>• PRESERVE_REGISTER_SYNONLY ON</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• preserve_syn_only or syn_preservesyn_only (synthesis attributes)</td>
</tr>
<tr>
<td>Preserve Fan-Out Free Register Node</td>
<td>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: (<em>noprune</em>)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 set the</td>
<td>Yes</td>
<td>• PRESERVEREGISTER_FANOUT_FREE_NODE ON</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• no_prune on (synthesis attribute)</td>
</tr>
</tbody>
</table>
## Assignment Description

<table>
<thead>
<tr>
<th>Assignment</th>
<th>Description</th>
<th>Allows Fitter Optimization?</th>
<th>Assignment Syntax</th>
</tr>
</thead>
</table>
| PRESERVE_FANOUT_FREE_NODE         | 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; | Yes | *
| 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  
preserver, syn_perserve, or keep on (synthesis attributes)

### 1.2.2. 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.

### 1.3. 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.

**Related Information**

- Running the Fitter on page 11
- Viewing Fitter Reports on page 13

### 1.3.1. Using the Compilation Dashboard

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

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.
Figure 4. **Compilation Dashboard**

- Click the **Pencil** icon to edit settings for that stage of the compilation flow.
- Click any Compiler stage to run one or more Compiler stage.
- Click the **Report**, **RTL Viewer**, **Technology Map Viewer**, **Timing Analyzer**, or **Snapshot Viewer** icons for analysis of stage results.

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.

**Related Information**

Analyzing Compiler Snapshots on page 21

### 1.3.2. 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.

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. 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.

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.

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 62 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**(2).
   - 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 31

### 1.3.2.1. Fitter Commands

Launch Fitter processes from the Processing menu or Compilation Dashboard with Fitter commands.

<table>
<thead>
<tr>
<th>Table 4. Start Fitter Commands</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Command</strong></td>
</tr>
<tr>
<td>Start Fitter (Plan)</td>
</tr>
<tr>
<td>Start Fitter (Place)</td>
</tr>
<tr>
<td>Start Fitter (Route)</td>
</tr>
<tr>
<td>Start Fitter (Retime)</td>
</tr>
<tr>
<td>Start Fitter (Finalize)</td>
</tr>
</tbody>
</table>

---

(2) Retime available for Intel Hyperflex™ architecture devices only.
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.

1.3.2.2. 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.

1.3.3. 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.

1.3.3.1. 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 5. Plan Stage Reports (Intel Arria 10 and Intel Cyclone 10 GX Designs)

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 and Intel Agilex designs.

1.3.3.2. 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.
1.3.3.2.1. Global Signal Visualization Report

For Intel Stratix 10 and Intel Agilex designs, you can access the Global Signal Visualization report to view global signal routing and clock sector utilization in an interactive heat-map. This report allows you to track the routing and placement of each individual clock. You can use this data to analyze global signal routing congestion issues, and to debug global signal placement and routing failures.

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.

The interactive heatmap color gradients show clock sector congestion of the clock signals terminating inside the sector. Hover the cursor over a clock signal in the table to highlight the clocks sectors and routing elements. Select a clock signal in the table to dim all irrelevant sectors and routing elements, while highlighting only the clock’s sectors and routing elements. The global clock signal routing on different layers displays in the report’s stacked layer view.
Figure 7. **Global Routing Wire Utilization (Single Layer)**

![Single Layer Routing vs Multi-Layer Routing](image-url)

Figure 8. **Heat Map Sector and Routing Wire Utilization (All Layers)**

![Heat Map with Color Coding for Wire Utilization](image-url)
Filter the display to **Show Routing Utilization** and **Show Sector Utilization**. The content of the table changes based on the selections you make in the heatmap. You can search for **Signal Names**, and then select the signal names to display its properties in the lower pane. Select any signal to **Locate** in other tools.

**Figure 9. Signals Names and Property Details**

![Image of heat-map and signal list]

### 1.3.3.3. 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 10. Route Stage Reports**

![Image of route stage report]

### 1.3.3.3.1. Global Router Wire Utilization Map Report

The Global Router Wire Utilization Map report displays global signal routing in an interactive heat-map. This report shows routing utilization rate of long and short routing wires. You can also use this report to obtain a detailed view of all the nets in
the design. The report's heatmap grid shows the available device LABs. The color of the grid ranges from blue to red as the utilization rate changes from 0% to 100%. The color becomes pink if the utilization rate is greater than 100%.

Figure 11. Global Router Wire Utilization Heat Map (Multiple Layers)

Filter the Global Router Wire table to show short or long Wirelengths in all Directions. The content of the table changes based on the selections you make in the heatmap. You can search for Signal Names, and then single- or multi-select the signal names to display properties in the lower pane. Select one or more node in the table to Locate in various editors.

Figure 12. Global Router Wire Utilization Details
1.3.3.4. Retime Stage Reports

For Intel Stratix 10 and Intel Agilex designs, 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 13. Retiming Limit Details

1.3.3.5. Finalize Stage Reports

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

- HSLP Summary. For Intel Arria 10 and Intel Cyclone 10 GX designs, the Compiler converts unnecessary tiles to High-Speed or Low-Power (HSLP) tiles.
- Post-route hold fix-up data. For Intel Stratix 10 and Intel Agilex 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.
### 1.4. 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, you can 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.
Table 5. Incremental Optimization at Fitter Stages

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

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.

1.4.1. 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.
1.4.2. Analyzing Compiler Snapshots

You can analyze the results of a compilation snapshot to evaluate your design before running the next stage, or before running a full compilation. Analyze Compiler snapshots to isolate potential problems and reduce the overall time you spend running design compilation.

1.4.2.1. Running Snapshot Viewer

You can run the Snapshot Viewer to assist with timing closure and design analysis after running the Plan, Place, or Route stages of the Fitter. The Snapshot Viewer allows you to run various analysis tasks from the Flow Navigator to achieve faster timing closure and maximize design performance.
Figure 17. **Snapshot Viewer Flow Navigator**

<table>
<thead>
<tr>
<th>Design Task</th>
<th>Available at Snapshot</th>
<th>Snapshot Viewer Commands</th>
</tr>
</thead>
</table>
| Timing Closure—Analyze Failing Paths | Planned, Placed, Routed | • **List Top Failing Paths**—lists all top failing paths in **Snapshot Selections**. Select a path to locate in RTL Viewer or Chip Planner.  
• **Show Full Timing Path in the Schematic**—highlights the path in RTL Viewer for further analysis.  
• **Show Full Timing Path in the Chip View**—highlights the path in Chip Planner for further analysis.  
• **View Path Characteristics**—The path loads in the Timing Analyzer for further analysis. |
| Timing Closure—Analyze Clocking | Placed                | **Show Global Clock Visualization**—loads the Global Signal Visualization report for the snapshot that allows you to visualize clock sector utilization. |
| Timing Closure—Analyze High Fanout Nets | Placed, Routed       | • **List High Fanout Nets**—lists high fan-out nets in **Snapshot Selections**. Select a path to locate in RTL Viewer or Chip Planner.  
• **Show High Fanout Nets in the Schematic**—highlights the paths in RTL Viewer for further analysis.  
• **Show High Fanout Nets in the Chip View**—highlights the paths in Chip Planner for further analysis. |
| Timing Closure—Validate Constraints | Planned, Placed     | **Timing Exceptions**—the Tcl Console reports analysis of the project .sdc file. |
| Timing Closure—Analyze Congestion | Placed, Routed       | **Show Logic Lock Regions with Congestion Heat Map**—the Chip Planner displays the Logic Lock regions in a congestion heat map for further analysis. |

The following sections describe each analysis task in detail.

Analyzing Failing Paths with Snapshot Viewer on page 23
Analyzing Clocking with Snapshot Viewer on page 25
Analyzing High Fan-out Nets with Snapshot Viewer on page 26
1. To run the Plan, Place, or Route stage of the Fitter, double-click the stage in the Compilation Dashboard.

2. After the stage completes, click the **Snapshot Viewer** icon for that stage in the Compilation Dashboard. The Snapshot Viewer opens.

3. Under **Analyze Failing Paths**, click **List Top Failing Paths**.

4. In **Snapshot Selections**, select the failing path for analysis.
5. Under **Select Failing Path to Analyze**, click **Show Full Timing Path in the Chip View**. The path displays and highlights in the Chip Planner for further analysis.

6. Under **Select Failing Path to Analyze**, click **Show Full Timing Path in Schematic**. The path displays and highlights in RTL Viewer for further analysis.

7. Under **Select Failing Path to Analyze**, click **View Path Characteristics**. The path loads in the Timing Analyzer for further analysis.
1.4.2.1.2. Analyzing Clocking with Snapshot Viewer

1. Run the Place stage, and then click the **Snapshot Viewer** icon for the stage in the Compilation Dashboard. The Snapshot Viewer opens.

2. Under **Analyze Clocking**, double-click **Show Global Clock Visualization**. The Global Signal Visualization report displays in Snapshot Viewer for analysis of clock sector and routing utilization.

Figure 23. **Global Clock Visualization Report**
1.4.2.1.3. Analyzing High Fan-out Nets with Snapshot Viewer

1. To run the Place or Route stage of the Fitter, double-click the stage in the Compilation Dashboard.
2. After the stage completes, click the **Snapshot Viewer** icon for that stage in the Compilation Dashboard. The Snapshot Viewer opens.
3. Under **Analyze High Fanout Nets**, click **Show High Fanout Nets in the Schematic**. The path displays and highlights in Tech Map Viewer for further analysis.
4. Under **Analyze High Fanout Nets**, click **Show High Fanout Nets in the Chip View**. The path displays and highlights in the Chip Planner for further analysis.

**Figure 24. Non-Global High Fan-Out Signal in Chip Planner**

1.4.2.1.4. Validating Timing Constraints with Snapshot Viewer

1. To run the Plan or Place stage of the Fitter, double-click the stage in the Compilation Dashboard.
2. After the stage completes, click the Snapshot Viewer icon for that stage in the Compilation Dashboard. The Snapshot Viewer opens.
3. Under **Validate Constraints**, double-click **Timing Exceptions**. The Timing Exception Results report open, allowing additional analysis and locating to other tools.
1.4.2.1.5. Analyzing Congestion with Snapshot Viewer

1. To run the Place or Route stage of the Fitter, double-click the stage in the Compilation Dashboard.

2. After the stage completes, click the Snapshot Viewer icon for that stage in the Compilation Dashboard. The Snapshot Viewer opens.

3. Under Analyze Congestion, double-click Show Logic Lock Regions with Congestion Heat Map. The Chip Planner displays the Logic Lock regions in a congestion heat map for further analysis.
1.4.3. 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 26. Show Logic Lock Regions with Congestion Heat Map

Figure 27. Plan Stage Periphery Placement Message

Related Information
  For more information on RTL Viewer and Chip Planner
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 28. Input Pins Report**

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 and Intel Agilex designs.

**Figure 29. Global & Other Fast Signals Report Shows Clock Promotion (Intel Arria 10 and Intel Cyclone 10 GX FPGAs)**
1.5. Fast Forward Compilation Flow

The Intel Hyperflex architecture includes multiple Hyper-Registers in every routing segment and block input. Maximizing the use of Hyper-Registers improves the balance of time delays between registers, and mitigates critical path delays. Fast Forward compilation generates design recommendations to help you to break performance bottlenecks and maximize use of Hyper-Registers to drive the highest performance in Intel Stratix 10 and Intel Agilex designs.

![Hyper-Registers in Intel Hyperflex Architecture](image)

The Fast Forward compilation reports show precisely where to make the most impact with RTL changes, and the performance benefits you can expect from each change after removing retiming restrictions. The Fast Forward compilation flow includes the following high-level steps:

![Fast Forward Compile Flow](image)

1. Step 1: Run Register Retiming on page 31
2. Step 2: Review Retiming Results on page 31
3. Step 3: Run Fast Forward Compile on page 34
4. Step 4: Review Fast Forward Results on page 38
5. Step 5: Implement Fast Forward Recommendations on page 41
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 and Intel Agilex device interconnect.

The Fitter runs the Retime stage automatically following place and route when you target an Intel Stratix 10 or Intel Agilex 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. Click Retime on the Compilation Dashboard. The Compiler runs prerequisite stages automatically before running Retime stage.

2. Review the results of the register retiming stage, as Step 2: Review Retiming Results on page 31 describes.

1.5.2. Step 2: Review Retiming Results

Follow these steps to review the results of register retiming. Use the results to determine if additional performance improvements are necessary and possible by removing retiming limits.

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.
To further optimize, resolve any Limiting Reason in your design, and then rerun the Retime stage, as necessary.

### Table 7. Retiming Limit Details Report Data

<table>
<thead>
<tr>
<th>Report Data</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock Transfer</td>
<td>Lists each clock domain in your design. Click the domain to display data about each entry.</td>
</tr>
<tr>
<td>Limiting Reason</td>
<td>Specifies any design condition that prevent further register retiming improvement, such as any of the following conditions:</td>
</tr>
<tr>
<td></td>
<td>• <strong>Insufficient Registers</strong>—indicates insufficient quantity of registers at either end of the chain for retiming. Adding more registers can improve performance.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Short Path/Long Path</strong>—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 hyper-registers.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Path Limit</strong>—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.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Loops</strong>—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.</td>
</tr>
<tr>
<td>Critical Chain Details</td>
<td>Lists register timing path associated with the retiming limitations. Right-click any path to Locate Critical Chain in Technology Map Viewer.</td>
</tr>
</tbody>
</table>

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 as Step 3: Run Fast Forward Compile on page 34 describes.

### 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.


**Figure 34. Technology Map Viewer**

**Figure 35. Post-Fit Viewer After Retiming**

In the post-fit viewer, bypassed ALM registers are gray. Hyper-Registers are pink with the word "HYPER" below them. Used ALMs are pink without the word "HYPER" below them.
1.5.3. Step 3: Run Fast Forward Compile

Fast Forward compilation generates design-specific timing closure recommendations, and predicts the maximum performance after the removal of all timing restrictions.

You can review the Fast Forward recommendations and implement the changes in your RTL that remove timing restrictions and enable mobility within the netlist for register Hyper-Retiming.

You can run Fast Forward compilation for the entire design hierarchy, or for only specific instances in the hierarchy, as Fast Forward Compile By Hierarchy on page 35 describes.

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

1. Optionally, specify 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.

2. 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.
3. View timing closure recommendations in the Compilation Report to evaluate design performance and implement key RTL performance improvements, as Step 4: Review Fast Forward Results on page 38 describes.

1.5.3.1. Fast Forward Compile By Hierarchy

When enabled, Fast Forward compile runs on the entire design hierarchy by default. Optionally, you can specify the Enable Hyper-Retimer Fast Forward Hierarchy analysis during compilation assignment to include or exclude specific design subhierarchies and instances during Fast Forward compile. This technique allows you to focus Fast Forward reporting and optimization efforts on only specific areas of the design. Fast Forward compilation by hierarchy generates the same reports as Fast Forward compilation of the entire hierarchy.

Follow these steps to include or exclude specific design subhierarchies and instances during Fast Forward compilation:

1. To enable the optional Fast Forward Compilation stage during full compilation, turn on Fast Forward Timing Closure Recommendations on the Compilation Dashboard, or add the following assignment to the project .qsf:

```plaintext
set_global_assignment -name FLOW_ENABLE_HYPER RETIMER_FAST_FORWARD ON
```
2. To exclude a specific hierarchy or entity from Fast Forward Compilation, set the **Enable Hyper-Retimer Fast Forward Hierarchy analysis during compilation** assignment to **Off** in the Assignment Editor, or add the following assignment to the project `.qsf` for each hierarchy or entity that you want to exclude:

```
set_global_assignment -name HYPER_RETIMER_FAST_FORWARD_ON_HIERARCHY OFF <INSTANCE MODULE NAME>
```

3. To include a specific hierarchy or entity from Fast Forward Compilation, set the **Enable Hyper-Retimer Fast Forward Hierarchy analysis during compilation** assignment to **On** in the Assignment Editor, or add the following assignment to the project `.qsf` for each hierarchy or entity that you want to include:

```
set_global_assignment -name HYPER_RETIMER_FAST_FORWARD_ON_HIERARCHY ON <INSTANCE MODULE NAME>
```

4. Click the **Fast Forward Timing Closure Recommendations** stage on the Compilation Dashboard, or click **Processing ➤ Start Compilation** to run a full compilation that includes Fast Forward Compile.

You can mix **ON** and **OFF** assignments for the same instance within a single `.qsf`. If you assign mixed **ON** and **OFF** assignments to the same instance, the last assignment that appears in the `.qsf` takes precedence.

If you want to perform Fast Forward analysis for a subset of the hierarchies in your design, turn off Fast Forward analysis for all hierarchies that you want to omit from analysis. Otherwise, turn off Fast Forward analysis at the root hierarchy, and turn on Fast Forward analysis for the hierarchies that you want to analyze. The following examples show some of these assignment combinations, with respect to the **Example Design Hierarchy**.
Figure 38. Example Design Hierarchy

```plaintext
# This runs Fast Forward Compile on the entire hierarchy: A,B,C,D,E,F
# This produces the same result as if FAST_FORWARD_HIERARCHY was not set in the QSF
set_global_assignment -name FLOW_ENABLE_HYPER RETIMER FAST FORWARD ON
set_instance_assignment -name HYPER RETIMER FAST FORWARD ON_HIERARCHY ON -to |

# Runs Fast Forward Compile on B and E only, ignores A,C,D,F
set_global_assignment -name FLOW_ENABLE_HYPER RETIMER FAST FORWARD ON
set_instance_assignment -name HYPER RETIMER FAST FORWARD ON_HIERARCHY OFF -to |
set_instance_assignment -name HYPER RETIMER FAST FORWARD ON_HIERARCHY ON -to B

# Runs Fast Forward Compile on C only, ignores A,B,D,E,F
set_global_assignment -name FLOW_ENABLE_HYPER RETIMER FAST FORWARD ON
set_instance_assignment -name HYPER RETIMER FAST FORWARD ON_HIERARCHY OFF -to |
set_instance_assignment -name HYPER RETIMER FAST FORWARD ON_HIERARCHY OFF -to C
set_instance_assignment -name HYPER RETIMER FAST FORWARD ON_HIERARCHY OFF -to F

# ON instance HYPER RETIMER FAST FORWARD ON_HIERARCHY takes precedence
# If the assignments were reversed then FFC would not run
set_global_assignment -name FLOW_ENABLE_HYPER RETIMER FAST FORWARD ON
set_instance_assignment -name HYPER RETIMER FAST FORWARD ON_HIERARCHY OFF -to |
set_instance_assignment -name HYPER RETIMER FAST FORWARD ON_HIERARCHY OFF -to C
set_instance_assignment -name HYPER RETIMER FAST FORWARD ON_HIERARCHY OFF -to F

# This runs on A,B,C,F
set_global_assignment -name FLOW_ENABLE_HYPER RETIMER FAST FORWARD ON
set_instance_assignment -name HYPER RETIMER FAST FORWARD ON_HIERARCHY OFF -to D
set_instance_assignment -name HYPER RETIMER FAST FORWARD ON_HIERARCHY OFF -to E
```

Note: Instance assignments apply to the post-fit netlist. Therefore, you may need to define design partitions. Otherwise, instance names can change during synthesis, leading to unexpected assignment results. Refer to Creating a Design Partition on page 47.

1.5.3.2. HyperFlex Settings

The HyperFlex settings page controls whether Fast Forward Compilation analyzes and reports results for specific logical structures in the Intel Hyperflex architecture. 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 8. Advanced HyperFlex Settings

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fast Forward Compile Asynchronous Clears</td>
<td>Specifies how Fast Forward analysis accounts for registers with asynchronous clear signals. The options are:</td>
</tr>
<tr>
<td></td>
<td>• 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.</td>
</tr>
<tr>
<td></td>
<td>• Preserve—the Compiler never assumes removal or conversion of asynchronous clears for Fast Forward analysis.</td>
</tr>
<tr>
<td>Fast Forward Compile Cut All Clock Transfers</td>
<td>Cuts all clock transfers in Fast Forward Compilation analysis.</td>
</tr>
<tr>
<td>Fast Forward Compile Fully Registered DSP Blocks</td>
<td>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.</td>
</tr>
<tr>
<td>Fast Forward Compile Fully Registered RAM Blocks</td>
<td>Specifies how Fast Forward analysis accounts for RAM blocks that limit performance. Enable this option to analyze the blocks as fully registered.</td>
</tr>
<tr>
<td>Fast Forward Compile Maximum Additional Pipeline Stages</td>
<td>Specifies the maximum number of pipeline stages that Fast Forward compilation explores.</td>
</tr>
<tr>
<td>Fast Forward Compile User Preserve Directives</td>
<td>Specifies how Fast Forward compilation accounts for restrictions from user-preserve directives.</td>
</tr>
</tbody>
</table>

1.5.4. Step 4: Review Fast Forward 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. The following section describes these reports.

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 39. Current and Potential Performance in Clock Fmax Summary

Predicts Optimized Performance After Hyper-Retiming, Hyper-Pipelining, and Hyper-Optimization

<table>
<thead>
<tr>
<th>Clock Fmax Summary</th>
<th>Clock Name</th>
<th>Fmax Achieved with Hyper-Retiming</th>
<th>Fmax Achieved with Hyper-Pipelining</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>clock</td>
<td>91 MHz</td>
<td>91 MHz</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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 in the report to view the implementation details and performance calculations for that step.

Figure 40. Fast-Forward Details Report

Figure 41. Recommendations for Critical Chain
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 42. Locate Critical Chain in Fast Forward Viewer**

<table>
<thead>
<tr>
<th>Critical Chain at Fast Forward Limit</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Path Info</strong></td>
</tr>
<tr>
<td>Retiming Restriction</td>
</tr>
<tr>
<td>Long Path 1</td>
</tr>
<tr>
<td>Long Path 2</td>
</tr>
<tr>
<td>Long Path 3</td>
</tr>
<tr>
<td>Long Path 4</td>
</tr>
<tr>
<td>Long Path 5</td>
</tr>
<tr>
<td>Long Path 6</td>
</tr>
<tr>
<td>Locate in Fast Forward Viewer</td>
</tr>
</tbody>
</table>

**Figure 43. Fast Forward Viewer Shows Predictive Results**
Table 9. Fast Forward Details Report Data

<table>
<thead>
<tr>
<th>Report Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step</td>
<td>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.</td>
</tr>
<tr>
<td>Fast Forward Optimizations Analyzed</td>
<td>Summarizes the optimizations necessary to implement each optimization step.</td>
</tr>
<tr>
<td>Estimated $f_{MAX}$</td>
<td>Specifies the potential $f_{MAX}$ performance if you implement all Fast Forward optimization steps.</td>
</tr>
<tr>
<td>Optimizations Analyzed For Fast Forward Step</td>
<td>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.</td>
</tr>
<tr>
<td>Optimizations Analyzed (Cumulative)</td>
<td>Accumulated list of all design changes necessary to reach the selected Step.</td>
</tr>
<tr>
<td>Critical Chain at Fast Forward Limit</td>
<td>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.</td>
</tr>
<tr>
<td>Recommendations for Critical Chain</td>
<td>Lists register timing path associated with the retiming limitations. Right-click any path to Locate Critical Chain in Fast Forward Viewer.</td>
</tr>
</tbody>
</table>

1.5.5. Step 5: Implement Fast Forward Recommendations

Implement the Fast Forward timing closure recommendations in your design RTL and rerun synthesis and the Retime stage to perform Hyper-Retiming and 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. For more information, refer to Retiming Restrictions and Workarounds on page 42.

1. Implement one or more Fast Forward recommendations in your design RTL, 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 combinatorial 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.

2. Implement your RTL changes and perform Hyper-Retiming by re-running the Retime stage on the Compilation Dashboard (which also reruns prerequisite synthesis and fitting stages).
1.5.6. 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 $f_{\text{MAX}}$ 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 44 on page 42, 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.

![Sample Critical Chain](image)

Figure 44. 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 10. Hyper-Register Support for Various Design Conditions

<table>
<thead>
<tr>
<th>Design Condition</th>
<th>Hyper-Register Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial conditions that cannot be preserved</td>
<td>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.</td>
</tr>
<tr>
<td>Register has an asynchronous clear</td>
<td>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.</td>
</tr>
<tr>
<td>Register drives an asynchronous signal</td>
<td>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.</td>
</tr>
<tr>
<td>Register has don’t touch or preserve attributes</td>
<td>The Compiler does not retime registers with these attributes. If you use the <code>preserve</code> attribute to manage register duplication for high fan-out signals, try removing the <code>preserve</code> 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 <code>dont_merge</code> attribute. The Compiler retimes registers in ALMs, DDIOs, single port RAMs, and DSP blocks.</td>
</tr>
<tr>
<td>Register is a clock source</td>
<td>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.</td>
</tr>
<tr>
<td>Register is a partition boundary</td>
<td>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.</td>
</tr>
<tr>
<td>Register is a block type modified by an ECO operation</td>
<td>This restriction is uncommon. Avoid the restriction by making the functional change in the design source and recompiling, rather than performing an ECO.</td>
</tr>
<tr>
<td>Register location is an unknown block</td>
<td>This restriction is uncommon. You can often work around this condition by adding extra registers adjacent to the specified block type.</td>
</tr>
<tr>
<td>Register is described in the RTL as a latch</td>
<td>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.</td>
</tr>
<tr>
<td>Register location is at an I/O boundary</td>
<td>All designs contain I/O, but you can add additional pipeline stages next to the I/O boundary for Hyper-Retiming.</td>
</tr>
<tr>
<td>Combinational node is fed by a special source</td>
<td>This condition is uncommon, especially for performance-critical parts of a design.</td>
</tr>
<tr>
<td>Register is driven by a locally routed clock</td>
<td>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.</td>
</tr>
<tr>
<td>Register is a timing exception end-point</td>
<td>The Compiler does not retime registers that are sources or destinations of .sdc constraints.</td>
</tr>
<tr>
<td>Register with inverted input or output</td>
<td>This condition is uncommon.</td>
</tr>
<tr>
<td>Register is part of a synchronizer chain</td>
<td>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. Alternatively, you can reduce the detection</td>
</tr>
</tbody>
</table>

*continued...*
### 1. Design Condition

<table>
<thead>
<tr>
<th>Design Condition</th>
<th>Hyper-Register Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>number for that particular synchronizer chain Synchro-Register Chain Length</td>
<td>(default is 3). In some cases a synchronizer chain isn’t necessary, and shouldn’t be inferred.</td>
</tr>
<tr>
<td>Register with multiple period requirements for paths that start or end at the register (cross-clock boundary)</td>
<td>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.</td>
</tr>
</tbody>
</table>

### 1.6. Full Compilation Flow

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 62 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

1.7. 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 11. Exporting Compilation Results

<table>
<thead>
<tr>
<th>To Export Compilation Results For</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complete Design</td>
<td>Click <strong>Project ➤ Export Design</strong></td>
<td>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.</td>
</tr>
<tr>
<td>Design Partition</td>
<td>Click <strong>Project ➤ Export Design Partition</strong></td>
<td>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.</td>
</tr>
</tbody>
</table>

1.7.1. 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**.
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.

1.7.2. 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 45 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**

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 that contains the final snapshot, start compilation with Timing Analysis (Signoff) (**Processing ➤ Start ➤ Start Timing Analysis (Signoff)**).
1. Design Compilation

1.7.3. 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 47. 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:
   - 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.
4. To view and edit all design partitions in the project, click **Assignments ➤ Design Partitions Window**.

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

**Table 12. Design Partition Settings**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Partition Name</strong></td>
<td>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 () &quot;root_partition&quot; for each project revision.</td>
</tr>
<tr>
<td><strong>Hierarchy Path</strong></td>
<td>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 .</td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td>Double-click to specify one of the following partition types that control how the Compiler processes and implements the partition:</td>
</tr>
<tr>
<td></td>
<td>• <strong>Default</strong>—Identifies a standard partition. The Compiler processes the partition using the associated design source files.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Reconfigurable</strong>—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.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Reserved Core</strong>—Identifies a partition in a block-based design flow that is reserved for core development by a Consumer reusing the device periphery.</td>
</tr>
<tr>
<td><strong>Preservation Level</strong></td>
<td>Specifies one of the following preservation levels for the partition: \ continued... \</td>
</tr>
</tbody>
</table>
### 1.7.4. 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 47 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:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Not Set</strong></td>
<td>Specifies no preservation level. The partition compiles from source files.</td>
</tr>
<tr>
<td><strong>synthesized</strong></td>
<td>The partition compiles using the synthesized snapshot.</td>
</tr>
<tr>
<td><strong>final</strong></td>
<td>The partition compiles using the final snapshot.</td>
</tr>
<tr>
<td></td>
<td>With Preservation Level of synthesized or final, changes to the source code do not appear in the synthesis.</td>
</tr>
</tbody>
</table>

**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.
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 47 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.
1.7.5. 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 .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 49 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.
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.

1.7.6. 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.

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]
   ```

1.7.6.1. QDB File Attribute Types

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

**Table 13. QDB File Attributes**

<table>
<thead>
<tr>
<th>QDB Attribute Types</th>
<th>Attribute</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Information</td>
<td>Contents</td>
<td>Partition</td>
</tr>
<tr>
<td></td>
<td>Date</td>
<td>Thu Jan 23 10:56:23 2018</td>
</tr>
<tr>
<td></td>
<td>Device</td>
<td>10AX016C3U19E2LG</td>
</tr>
<tr>
<td></td>
<td>Entity (if Partition)</td>
<td>Counter</td>
</tr>
<tr>
<td></td>
<td>Family</td>
<td>Arria 10</td>
</tr>
<tr>
<td></td>
<td>Partition Name</td>
<td>root_partition</td>
</tr>
<tr>
<td></td>
<td>Revision Name</td>
<td>Top</td>
</tr>
<tr>
<td></td>
<td>Revision Type</td>
<td>PR_BASE</td>
</tr>
<tr>
<td></td>
<td>Snapshot</td>
<td>synthesized</td>
</tr>
<tr>
<td></td>
<td>Version</td>
<td>18.1.0 Pro Edition</td>
</tr>
<tr>
<td></td>
<td>Version-Compatable</td>
<td>Yes</td>
</tr>
</tbody>
</table>
| Resource Utilization (exported for partition QDB only) | 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
... |
1.7.7. 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.

![Clean Project Dialog Box Cleans the Project Database](image)

**Figure 54. Clean Project Dialog Box Cleans the Project Database**

1.8. Integrating Other EDA Tools

You can optionally integrate supported EDA synthesis, netlist partitioning, simulation, and signal integrity verification tools into the Intel Quartus Prime design flow.

The Intel Quartus Prime software supports input netlist files from supported EDA synthesis tools. The Compiler’s EDA Netlist Writer module (*quartus_eda*) can automatically generate output files for processing in other EDA tools. The EDA Netlist Writer runs optionally as part of a full compilation, or you can run EDA Netlist Writer separately from the GUI or at the command line. The following functions are available to simplify EDA tool integration:

<table>
<thead>
<tr>
<th>EDA Integration Task</th>
<th>EDA Integration Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specify settings for generation of output files for processing in other EDA tools.</td>
<td>Click <strong>Assignments ➤ Settings ➤ EDA Tool Settings</strong> to specify options for supported tools.</td>
</tr>
<tr>
<td>Generate output files for processing in other EDA tools.</td>
<td>Click <strong>Processing ➤ Start ➤ Start EDA Netlist Writer</strong> (or run <code>quartus_eda</code>) to generate files.</td>
</tr>
<tr>
<td>Compile RTL and gate-level simulation model libraries for your device, supported EDA simulators, and design language.</td>
<td>Click <strong>Tools ➤ Launch Simulation Library Compiler</strong> to compile simulation libraries easily.</td>
</tr>
</tbody>
</table>

*continued...*
### EDA Integration Task

Generate EDA tool-specific setup scripts to compile, elaborate, and simulate Intel FPGA IP models and simulation model library files.

### EDA Integration Function

Specify options for Simulation file output when generating Intel FPGA IP with IP parameter editor.

Generate files that allow supported EDA tools to perform netlist modifications, such as adding new modules, partitioning the netlist, and changing module connectivity.

Use the `quartus_eda -resynthesis` command to generate a Verilog Quartus Mapping File (.vqm) that contains a node-level (or atom) representation of the netlist in standard structural Verilog RTL.

Include files generated by other EDA design entry or synthesis tools in your project as synthesized design files.

Click Project ➤ Add/Remove Files In Project to add supported Design File files from other EDA tools.

### Related Information


#### 1.8.1. Generating a VQM Netlist for other EDA Tools

The EDA Netlist Writer (`quartus_eda`) can generate a node-level netlist in Verilog Quartus Mapping File (.vqm) format for use in other EDA tools. You can process the .vqm netlist in other EDA tools to add new modules, partition the netlist, or change connectivity. After third-party tool changes, you resynthesize and compile the .vqm in the Intel Quartus Prime software.

The .vqm format is standard structural Verilog RTL. The modules can be any Intel FPGA family-specific WYSIWYG type for core logic (such as, flip-flop, LUT, DSP, M20K). EDA Netlist Writer does not support .vqm output for periphery modules (such as transceivers, memory interfaces, I/O, or IP including these). The RTL is a fully flattened representation of the entire design hierarchy or partition. The module names capture the original hierarchy, although some renaming can occur to legalize names. There is no truncation of the netlist module names.

To perform .vqm netlist partitioning in other EDA tools, define a design partition that includes only core logic elements. Generate the partition netlist as step 3 on page 56 describes. After processing the .vqm in third-party tools, resynthesize the .vqm files either independently or as a design partition. If including a black box module instantiation in the .vqm, make connections between existing logic in the .vqm and the black box. Prior to resynthesis, specify the source file (.ip, .v, or .vqm) for the black box in the project .qsf.

### Table 15. VQM Netlist Generation Requirements and Limitations

<table>
<thead>
<tr>
<th>Requirement or Limitation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design partitions must only include core logic.</td>
<td>Design partitions must include only flip-flops, LUTs, DSPs, and on-chip memory. The EDA Netlist Writer does not support .vqm output for periphery modules (such as transceivers, memory interfaces, I/O, or IP that includes these).</td>
</tr>
<tr>
<td>Analysis &amp; Synthesis does not support some special characters in instance names that are legal in SystemVerilog.</td>
<td>Analysis &amp; Synthesis resolves these characters by placing the standard escape character <code>\</code> to escape the special character present in the RTL. If any of the hierarchical constraints (for example, SDC timing constraints) explicitly reference such a special character, modify these characters manually.</td>
</tr>
</tbody>
</table>

continued...
To generate a .vqm for processing in other EDA tools, follow these steps:

1. In the Intel Quartus Prime software, click Processing ➤ Start ➤ Start Analysis & Synthesis (or run quartus_syn) to synthesize the design netlist.

2. Create a design partition containing only core logic elements for the .vqm, as Creating a Design Partition on page 47 describes.

3. To generate the .vqm in the resynthesis directory, run any of the following commands at the command prompt:
   - To write out the entire design netlist to .vqm:
     
     ```sh
     quartus_eda -resynthesis=on <project_name>
     ```
   - To write out only a specific design partition netlist to .vqm:
     
     ```sh
     quartus_eda -resynthesis=on -partition=<name> <project_name>
     ```
   - To write out any sub partition as a black-box netlist to .vqm:
     
     ```sh
     quartus_eda -resynthesis=on -exclude_sub_partitions <project_name>
     ```

     You can also combine -exclude_sub_partitions with -partition.

4. View the resulting .vqm in the resynthesis directory, and specify the .vqm as input to your EDA tool.

5. After processing the .vqm in another EDA tool, add the .vqm as an Intel Quartus Prime project design file by clicking Project ➤ Add/Remove Files In Project. Avoid module or entity name collisions, as Table 15 on page 55 describes.

6. Run Analysis & Synthesis on the project, followed by the remaining Compiler stages.

### 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:
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


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.
Table 16. Verilog HDL Input Settings

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Verilog Version</td>
<td>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.</td>
</tr>
<tr>
<td>Library Mapping File</td>
<td>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.</td>
</tr>
<tr>
<td>Verilog HDL Macro</td>
<td>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.</td>
</tr>
</tbody>
</table>

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**

```verilog
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**

```verilog
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**

```verilog
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**

```plaintext
@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:

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

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

```plaintext
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:

```bash
--write_settings_files=off
```

### 1.9.2. VHDL Synthesis Support

Intel Quartus Prime synthesis supports the following VHDL language standards.


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 17. VHDL Input Settings

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VHDL Version</td>
<td>Specifies the VHDL standard for use during synthesis of VHDL input design files. Select the language standards that corresponds with the VHDL files.</td>
</tr>
<tr>
<td>Library Mapping File</td>
<td>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.</td>
</tr>
</tbody>
</table>

Figure 56. VHDL Input Settings Dialog Box

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

```vhdl
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. Compiler Optimization Techniques

You can apply various optimization techniques via settings and entity assignments to achieve your design requirements during compilation. For example, you can specify options to preserve specific registers through synthesis processing, apply fractal synthesis, enable register retiming, and various other targeted Compiler optimizations.

1.10.1. 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>
<thead>
<tr>
<th>Optimization Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Balanced (normal flow)</td>
<td>The Compiler optimizes synthesis for balanced implementation that respects timing constraints.</td>
</tr>
<tr>
<td>High Performance Effort</td>
<td>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.</td>
</tr>
<tr>
<td>High Performance with Maximum Placement Effort</td>
<td>Enables the same Compiler optimizations as High Performance Effort, with additional placement optimization effort.</td>
</tr>
<tr>
<td>Superior Performance</td>
<td>Enables the same Compiler optimizations as High Performance Effort, and adds more optimizations during Analysis &amp; 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.</td>
</tr>
<tr>
<td>Superior Performance with Maximum Placement Effort</td>
<td>Enables the same Compiler optimizations as Superior Performance, with additional placement optimization effort.</td>
</tr>
</tbody>
</table>

continued...
### Optimization Mode and Description

<table>
<thead>
<tr>
<th>Optimization Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Aggressive Area</strong></td>
<td>The Compiler makes aggressive effort to reduce the device area required to implement the design at the potential expense of design performance.</td>
</tr>
<tr>
<td><strong>High Placement Routability Effort</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>High Packing Routability Effort</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Optimize Netlist for Routability</strong></td>
<td>The Compiler implements netlist modifications to increase routability at the possible expense of performance.</td>
</tr>
<tr>
<td><strong>High Power Effort</strong></td>
<td>The Compiler makes high effort to optimize synthesis for low power. High Power Effort increases synthesis run time.</td>
</tr>
<tr>
<td><strong>Aggressive Power</strong></td>
<td>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.</td>
</tr>
</tbody>
</table>
| **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. |

**1.10.2. Allow Register Retiming**

The Allow Register Retiming option on the Register Optimization tab 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|”
```
1.10.3. Automatic Gated Clock Conversion

Clock gating saves power in ASIC designs by adding more logic to a circuit to prune the clock tree. Pruning the clock tree disables portions of the circuitry so that the flip-flops are not required to switch states. When using an Intel Quartus Prime FPGA to prototype ASIC designs, you must convert clock gates to clock enables in your design.

Table 19. Gated Clock Conversion Example

<table>
<thead>
<tr>
<th>ASIC Gated Clock Example</th>
<th>FPGA Clock Enable Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>module infer_enable (clk, reset, d, en, q);</td>
<td>module infer_enable (clk, reset, d, en, q);</td>
</tr>
<tr>
<td>input d, en, clk, reset;</td>
<td>input d, en, clk, reset;</td>
</tr>
<tr>
<td>output q;</td>
<td>output q;</td>
</tr>
<tr>
<td>wire gated_clk;</td>
<td>reg q;</td>
</tr>
<tr>
<td>reg q;</td>
<td>always@(posedge clk or reset)</td>
</tr>
<tr>
<td>assign gated_clk = clk &amp; en;</td>
<td>begin</td>
</tr>
<tr>
<td>always@(posedge gated_clk or reset)</td>
<td>if (!reset)</td>
</tr>
<tr>
<td>begin</td>
<td>q &lt;= 1'b0;</td>
</tr>
<tr>
<td>if (!reset)</td>
<td>else if (en)</td>
</tr>
<tr>
<td>else</td>
<td>q &lt;= q;</td>
</tr>
<tr>
<td>q &lt;= d;</td>
<td>end</td>
</tr>
<tr>
<td>endmodule</td>
<td>end</td>
</tr>
</tbody>
</table>

Rather than manually converting gated clocks in your RTL, you can specify the **Auto Gated Clock Conversion** setting to automatically convert gated base clocks in the design to clock enables. You can apply this setting globally to all gated base clocks in the design, or to one or more specific clock signals.

Table 20. Gated Clock Conversion Settings

<table>
<thead>
<tr>
<th>Setting Scope</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global</td>
<td>Enable the <strong>Auto Gated Clock Conversion</strong> option at <strong>Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Synthesis)</strong>. Alternatively, add the global assignment to the project .qsf:</td>
</tr>
<tr>
<td></td>
<td>set_global_assignment -name SYNTH_GATED_CLOCK_CONVERSION on</td>
</tr>
<tr>
<td>Instance-specific</td>
<td>Specify the <strong>Auto Gated Clock Conversion</strong> for one or more instances in the Assignment Editor (Assignments ➤ Assignment Editor). Alternatively, add the instance assignment to the project .qsf:</td>
</tr>
<tr>
<td></td>
<td>set_instance_assignment -name SYNTH_GATED_CLOCK_CONVERSION on -to clk_in</td>
</tr>
</tbody>
</table>

Following design synthesis, view the results of gated clock conversion in the Gated Clock Conversion Details report. The report lists all converted and unconverted gated clocks with their base clocks. For unconverted gated clocks, the report specifies the reason the clock is not converted.

**Note:** Automatic gated clock conversion supports explicit RAMs (such as WYSIWYG RAMs and Intel FPGA memory IP), but does not support inferred RAMs.
1.10.4. 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.

1.10.5. Fast Preserve Option

Enabling the Fast Preserve option on the Incremental Compile tab specifies that the Compiler can simplify a preserved partition to only interface logic.

Interface logic is logic at the partition boundary that interfaces with the rest of the design.

1.10.6. Fractal Synthesis Optimization

Fractal synthesis optimizations can be 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.

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). You can enable fractal synthesis globally or for specific multipliers, as Enabling or Disabling Fractal Synthesis on page 70 describes.

Project-Wide Fractal Synthesis Considerations

Note: Fractal synthesis optimization is most suitable for designs with deep-learning accelerators or other high-throughput, arithmetic-intensive functions that exceed all DSP resources. Enabling fractal synthesis project-wide can cause unnecessary bloat on modules that are not suitable for fractal optimizations. Consider the following factors before enabling fractal synthesis optimization project-wide:
• Intel FPGA 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 global 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 project-wide 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 the Compiler is packing computational logic into a smaller number of LABs. If the design is already highly utilized, the Compiler can skip this stage.

— 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.

<table>
<thead>
<tr>
<th>Table 21. Fractal Synthesis Area Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Device</strong></td>
</tr>
<tr>
<td>Intel Arria 10 and Intel Cyclone 10 GX</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Intel Stratix 10 and Intel Agilex Devices</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

**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 58. Multiplier Retiming

Before Multiplier Retiming

After 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

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

```verilog
(* 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));

```
Intel Quartus Prime synthesis prints the following messages to the console:

**Figure 59. Console Messages**

<table>
<thead>
<tr>
<th>Message</th>
<th>Message ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inferred optimized multiplier from the following logic: &quot;core1\mult_2&quot;</td>
<td>20192</td>
</tr>
<tr>
<td>Inferred optimized multiplier from the following logic: &quot;core1\mult_1&quot;</td>
<td>20192</td>
</tr>
<tr>
<td>Inferred optimized multiplier from the following logic: &quot;core2\mult_2&quot;</td>
<td>20192</td>
</tr>
<tr>
<td>Inferred optimized multiplier from the following logic: &quot;core2\mult_1&quot;</td>
<td>20192</td>
</tr>
<tr>
<td>Inferred optimized multiplier from the following logic: &quot;core2\mult_0&quot;</td>
<td>20192</td>
</tr>
<tr>
<td>Applied dense packing to &quot;core2&quot;. Area: 6 LABs. Logic density: 0.941567.</td>
<td>20193</td>
</tr>
<tr>
<td>Applied dense packing to &quot;core1&quot;. Area: 6 LABs. Logic density: 0.941567.</td>
<td>20193</td>
</tr>
</tbody>
</table>

In the Chip Planner, you can observe this design having two unsigned dot-product cores. These cores are independently optimized and placed. The LAB resources are nearly 100% optimized, as the following image shows:

**Figure 60. Design Placement**

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

```vhdl
(* 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

Intel Quartus Prime synthesis displays the following messages in the console:

**Figure 61. Console Messages**

<table>
<thead>
<tr>
<th>Message</th>
<th>Message ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inferred optimized multiplier from the following logic: &quot;core1*mult_2&quot;</td>
<td>20152</td>
</tr>
<tr>
<td>Inferred optimized multiplier from the following logic: &quot;core1*mult_1&quot;</td>
<td>20152</td>
</tr>
<tr>
<td>Inferred optimized multiplier from the following logic: &quot;core1*mult_0&quot;</td>
<td>20152</td>
</tr>
<tr>
<td>Inferred optimized multiplier from the following logic: &quot;core2*mult_3&quot;</td>
<td>20152</td>
</tr>
<tr>
<td>Inferred optimized multiplier from the following logic: &quot;core2*mult_2&quot;</td>
<td>20152</td>
</tr>
<tr>
<td>Inferred optimized multiplier from the following logic: &quot;core2*mult_1&quot;</td>
<td>20152</td>
</tr>
<tr>
<td>Inferred optimized multiplier from the following logic: &quot;core2*mult_0&quot;</td>
<td>20152</td>
</tr>
<tr>
<td>Timing-Driven Synthesis is running</td>
<td>286530</td>
</tr>
<tr>
<td>Applied dense packing to &quot;core2&quot;. Area: 6 LABs. Logic density: 0.95.</td>
<td>20193</td>
</tr>
<tr>
<td>Applied dense packing to &quot;core2&quot;. Area: 6 LABs. Logic density: 0.95.</td>
<td>20193</td>
</tr>
</tbody>
</table>

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

**Figure 62. 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 63. Continuous Arithmetic Packing**

![Continuous Arithmetic Packing](image)

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.

### 1.10.6.1. Enabling or Disabling Fractal Synthesis

For Intel Stratix 10 and Intel Agilex devices, fractal synthesis optimization runs automatically for small multipliers (any A*B statement in Verilog HDL or VHDL where bit-width of the operands is 7 or less). You can also disable automatic fractal synthesis for small multipliers for these devices using either of the following methods:

- In RTL, set the DSP multstyle, as "Multstyle Verilog HDL Synthesis Attribute" describes. For example:

  ```vhd
  (* multstyle = "dsp" *) module foo(...);
  module foo(...) /* synthesis multstyle = "dsp" */;
  ```

- In the .qsf file, add as an assignment as follows:

  ```qsf
  set_instance_assignment -name DSP_BLOCK_BALANCING_IMPLEMENTATION \ DSP_BLOCKS -to r
  ```
In addition, for Intel Stratix 10, Intel Agilex, Intel Arria 10, and Intel Cyclone 10 GX devices, you can enable fractal synthesis globally or for specific multipliers with the Fractal Synthesis GUI option or the corresponding FRACTAL_SYNTHESIS .qsf assignment:

- 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>
  ```

In the user interface, follow these steps:

1. Click Assignments ➤ Assignment Editor.
2. 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 64. Fractal Synthesis Assignment in Assignment Editor**

**Related Information**
Multstyle Verilog HDL Synthesis Attribute
In Intel Quartus Prime Help.

**1.11. Synthesis Settings Reference**

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

**1.11.1. 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 22. Advanced Synthesis Settings (1 of 13)

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allow Any RAM Size for Recognition</td>
<td>Allows the Compiler to infer RAMs of any size, even if the RAMs do not meet the current minimum requirements.</td>
</tr>
<tr>
<td>Allow Any ROM Size for Recognition</td>
<td>Allows the Compiler to infer ROMs of any size even if the ROMs do not meet the design's current minimum size requirements.</td>
</tr>
<tr>
<td>Allow Any Shift Register Size for Recognition</td>
<td>Allows the Compiler to infer shift registers of any size even if they do not meet the design’s current minimum size requirements.</td>
</tr>
<tr>
<td>Allow Register Duplication</td>
<td>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: Only available for Intel Arria 10 and Intel Cyclone 10 GX devices.</td>
</tr>
<tr>
<td>Allow Register Merging</td>
<td>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: Only available for Intel Arria 10 and Intel Cyclone 10 GX devices.</td>
</tr>
<tr>
<td>Allow Shift Register Merging Across Hierarchies</td>
<td>Allows the Compiler to take shift registers from different hierarchies of the design and put the registers in the same RAM.</td>
</tr>
<tr>
<td>Allow Synchronous Control Signals</td>
<td>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.</td>
</tr>
</tbody>
</table>

### Table 23. Advanced Synthesis Settings (2 of 13)

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analysis &amp; Synthesis Message Level</td>
<td>Specifies the type of Analysis &amp; Synthesis messages the Compiler display. <strong>Low</strong> displays only the most important Analysis &amp; Synthesis messages. <strong>Medium</strong> displays most messages, but hides the detailed messages. <strong>High</strong> displays all messages.</td>
</tr>
<tr>
<td>Auto Clock Enable Replacement</td>
<td>Allows the Compiler to locate logic that feeds a register and move the logic to the register's clock enable input port.</td>
</tr>
<tr>
<td>Auto DSP Block Replacement</td>
<td>Allows the Compiler to find a multiply-accumulate function or a multiply-add function that can be replaced with a DSP block.</td>
</tr>
<tr>
<td>Auto Gated Clock Conversion</td>
<td>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.</td>
</tr>
</tbody>
</table>

### Table 24. Advanced Synthesis Settings (3 of 13)

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Auto Open-Drain Pins</td>
<td>Allows the Compiler to automatically convert a tri-state buffer with a strong low data input into the equivalent open-drain buffer.</td>
</tr>
<tr>
<td>Auto RAM Replacement</td>
<td>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.</td>
</tr>
<tr>
<td>Auto ROM Replacement</td>
<td>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.</td>
</tr>
</tbody>
</table>

**continued...**
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Auto Resource Sharing</td>
<td>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&lt;sub&gt;MAX&lt;/sub&gt; of your design.</td>
</tr>
<tr>
<td>Auto Shift Register Placement</td>
<td>Allows the Compiler to find a group of shift registers of the same length that are replaceable with the altshift&lt;sub&gt;taps&lt;/sub&gt; 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.</td>
</tr>
<tr>
<td>Automatic Parallel Synthesis</td>
<td>Option to enable/disable automatic parallel synthesis. Use this option to speed up synthesis compile time by using multiple processors when available.</td>
</tr>
</tbody>
</table>

### Table 25. Advanced Synthesis Settings (4 of 13)

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block Design Naming</td>
<td>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.</td>
</tr>
<tr>
<td>Clock MUX Protection</td>
<td>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.</td>
</tr>
<tr>
<td>DSP Block Balancing</td>
<td>Allows you to control the conversion of certain DSP block slices during DSP block balancing.</td>
</tr>
</tbody>
</table>

### Table 26. Advanced Synthesis Settings (5 of 13)

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disable DSP Negate Inferencing</td>
<td>Allows you to specify whether to use the negate port on an inferred DSP block.</td>
</tr>
<tr>
<td>Disable Register Merging Across Hierarchies</td>
<td>Specifies whether the Compiler allows merging of registers that are in different hierarchies if their inputs are the same.</td>
</tr>
<tr>
<td>Enable Formal Verification Support</td>
<td>Enables the Compiler to write scripts for use with the OneSpin* formal verification tool.</td>
</tr>
<tr>
<td>Enable State Machines Inference</td>
<td>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.</td>
</tr>
<tr>
<td>Force Use of Synchronous Clear Signals</td>
<td>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.</td>
</tr>
<tr>
<td>Fractal Synthesis</td>
<td>Turning this option On directs the Compiler to apply dense packing to arithmetic blocks, minimizing the area of the design for arithmetic-intensive designs.</td>
</tr>
<tr>
<td>HDL Message Level</td>
<td>Specifies the type of HDL messages you want to view, including messages that display processing errors in the HDL source code. <strong>Level1</strong> displays only the most important HDL messages. <strong>Level2</strong> displays most HDL messages, including warning and information based messages. <strong>Level3</strong> displays all HDL messages, including warning and information based messages and alerts about potential design problems or lint errors.</td>
</tr>
</tbody>
</table>
### Table 27. Advanced Synthesis Settings (6 of 13)

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ignore GLOBAL Buffers</td>
<td>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.</td>
</tr>
<tr>
<td>Ignore LCELL Buffers</td>
<td>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.</td>
</tr>
<tr>
<td>Ignore Maximum Fan-Out Assignments</td>
<td>Directs the Compiler to ignore the Maximum Fan-Out Assignments on a node, an entity, or the whole design.</td>
</tr>
<tr>
<td>Ignore SOFT Buffers</td>
<td>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.</td>
</tr>
</tbody>
</table>

### Table 28. Advanced Synthesis Settings (7 of 13)

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ignore translate_off and synthesis_off Directives</td>
<td>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.</td>
</tr>
<tr>
<td>Infer RAMs from Raw Logic</td>
<td>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 altsynram 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.</td>
</tr>
<tr>
<td>Iteration Limit for Constant Verilog Loops</td>
<td>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.</td>
</tr>
<tr>
<td>Iteration Limit for non-Constant Verilog Loops</td>
<td>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.</td>
</tr>
</tbody>
</table>

### Table 29. Advanced Synthesis Settings (8 of 13)

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum DSP Block Usage</td>
<td>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.</td>
</tr>
<tr>
<td>Maximum Number of LABs</td>
<td>Specifies the maximum number of LABs that Analysis &amp; 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.</td>
</tr>
<tr>
<td>Maximum Number of M4K/M9K/M20K/M10K Memory Blocks</td>
<td>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.</td>
</tr>
</tbody>
</table>

### Table 30. Advanced Synthesis Settings (9 of 13)

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum Number of Registers Created from Uninferred RAMs</td>
<td>Specifies the maximum number of registers that Analysis &amp; 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...</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>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.</td>
<td></td>
</tr>
<tr>
<td>NOT Gate Push-Back</td>
<td>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.</td>
</tr>
<tr>
<td>Number of Inverted Registers Reported in Synthesis Report</td>
<td>Specifies the maximum number of inverted registers that the Synthesis report displays.</td>
</tr>
<tr>
<td>Number of Protected Registers Reported in Synthesis Report</td>
<td>Specifies the maximum number of protected registers that the Synthesis Report displays.</td>
</tr>
<tr>
<td>Number of Removed Registers Reported in Synthesis Migration Checks</td>
<td>Specifies the maximum number of rows that the Synthesis Migration Check report displays.</td>
</tr>
<tr>
<td>Number of Swept Nodes Reported in Synthesis Report</td>
<td>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.</td>
</tr>
<tr>
<td>Number of Rows Reported in Synthesis Report</td>
<td>Specifies the maximum number of rows that the Synthesis report displays. Note: Only available for Intel Arria 10 and Intel Cyclone 10 GX devices.</td>
</tr>
<tr>
<td>Optimization Technique</td>
<td>Specifies an overall optimization goal for Analysis &amp; Synthesis. Specify a Balanced strategy, or optimize for Performance, Area, Routability, Power, or Compile Time. The Compiler targets the optimization goal you specify.</td>
</tr>
</tbody>
</table>

### Table 31. Advanced Synthesis Settings (10 of 13)

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perform WYSIWYG Primitive Resynthesis</td>
<td>Specifies whether to perform WYSIWYG primitive resynthesis during synthesis. This option uses the setting specified in the Optimization Technique logic option.</td>
</tr>
<tr>
<td>Power-Up Don’t Care</td>
<td>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.</td>
</tr>
<tr>
<td>Power Optimization During Synthesis</td>
<td>Controls the power-driven compilation setting of Analysis &amp; Synthesis. This option determines how aggressively Analysis &amp; 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.</td>
</tr>
</tbody>
</table>

### Table 32. Advanced Synthesis Settings (11 of 13)

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Remove Duplicate Registers</td>
<td>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...</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Option</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>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.</td>
<td></td>
</tr>
<tr>
<td>Remove Redundant Logic Cells</td>
<td>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.</td>
</tr>
<tr>
<td>Report Parameter Settings by Entity Instance</td>
<td>Specifies whether the Synthesis report includes the reports in the Parameter Settings by Entity Instance folder.</td>
</tr>
<tr>
<td>Report PR Initial Values as Errors</td>
<td>Allows you to flag explicitly defined initial values found in PR partitions as Errors instead of Warnings.</td>
</tr>
<tr>
<td>Report Source Assignments</td>
<td>Specifies whether the Synthesis report includes reports in the Source Assignments folder.</td>
</tr>
<tr>
<td>Resource Aware Inference for Block RAM</td>
<td>Specifies whether RAM, ROM, and shift-register inference should take the design and device resources into account.</td>
</tr>
</tbody>
</table>
| 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 ($t_{\text{MAX}}$).  
  • Off—disables multiplexer restructuring; it does not decrease logic element usage and does not affect design clock speed ($t_{\text{MAX}}$).  
  • 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 ($t_{\text{MAX}}$). |
| 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 or Intel Agilex 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 34. Advanced Synthesis Settings (13 of 13)

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>State Machine Processing</td>
<td>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.</td>
</tr>
<tr>
<td>Strict RAM Replacement</td>
<td>When this option is On, the Compiler replace RAM only if the hardware matches the design exactly.</td>
</tr>
<tr>
<td>Synchronization Register</td>
<td>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.</td>
</tr>
<tr>
<td>Chain Length</td>
<td></td>
</tr>
<tr>
<td>Synthesis Effort</td>
<td>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.</td>
</tr>
<tr>
<td>Synthesis Migration Check</td>
<td>Enables synthesis checks on Intel Arria 10 to Intel Stratix 10 design migration.</td>
</tr>
<tr>
<td>for Stratix 10</td>
<td></td>
</tr>
<tr>
<td>Timing-Driven Synthesis</td>
<td>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:</td>
</tr>
<tr>
<td></td>
<td>• Optimization Technique Speed—optimizes timing-critical portions of your design for performance at the cost of increasing area (logic and register utilization)</td>
</tr>
<tr>
<td></td>
<td>• Optimization Technique Balanced—also optimizes the timing-critical portions of your design for performance, but the option allows only limited area increase</td>
</tr>
<tr>
<td></td>
<td>• Optimization Technique Area—optimizes your design only for area</td>
</tr>
</tbody>
</table>

1.12. 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 35. Advanced Fitter Settings (1 of 8)

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALM Register Packing Effort</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>• 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.</td>
</tr>
<tr>
<td></td>
<td>• 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.</td>
</tr>
<tr>
<td></td>
<td>• 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.</td>
</tr>
<tr>
<td>Advanced Physical Synthesis</td>
<td>Enables the Physical Synthesis engine that includes combinational and sequential optimization during fitting to improve circuit performance.</td>
</tr>
<tr>
<td>Allow Delay Chains</td>
<td>Allows the Fitter to choose the optimal delay chain to meet tSU and tCO timing requirements for all I/O elements. Enabling this option may reduce the number of tSU violations, while introducing a minimal number of tCO violations. Enabling this option does not override delay chain settings on individual nodes.</td>
</tr>
</tbody>
</table>

continued...
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allow DSP Retiming</td>
<td>Allow retiming through DSP blocks.</td>
</tr>
<tr>
<td>Allow Early Global Retiming in the Fitter</td>
<td>Allows the Compiler to run global retiming early in the Fitter. Adam. 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.</td>
</tr>
<tr>
<td>Allow Hyper-Aware Register Chain Area Optimizations in the Fitter</td>
<td>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.</td>
</tr>
<tr>
<td>Allow RAM Retiming</td>
<td>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 retimer registers.</td>
</tr>
<tr>
<td>Allow Register Duplication</td>
<td>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 retimer registers.</td>
</tr>
<tr>
<td>Allow Register Merging</td>
<td>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 retimer registers.</td>
</tr>
<tr>
<td>Auto Delay Chains for High Fanout Input Pins</td>
<td>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 tSU violations, but the compile time increases significantly, as the Fitter tries to optimize the settings for all fan-outs.</td>
</tr>
<tr>
<td>Auto Fit Effort Desired Slack Margin</td>
<td>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.</td>
</tr>
</tbody>
</table>

Table 36. **Advanced Fitter Settings (2 of 8)**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Auto Global Clock</td>
<td>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.</td>
</tr>
<tr>
<td>Auto Global Register Control Signals</td>
<td>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.</td>
</tr>
<tr>
<td>Auto Packed Registers</td>
<td>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.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Auto RAM to MLAB Conversion</td>
<td>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.</td>
</tr>
<tr>
<td>Auto Register Duplication</td>
<td>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: Only available for Intel Arria 10 and Intel Cyclone 10 GX devices.</td>
</tr>
</tbody>
</table>

Table 37. Advanced Fitter Settings (3 of 8)

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable Auto-Pipelining</td>
<td>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: Only available for Intel Stratix 10 and Intel Agilex devices.</td>
</tr>
<tr>
<td>Enable Bus-Hold Circuitry</td>
<td>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.</td>
</tr>
<tr>
<td>Equivalent RAM and MLAB Paused Read Capabilities</td>
<td>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.</td>
</tr>
</tbody>
</table>

continued...
Table 38. Advanced Fitter Settings (4 of 8)

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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. |
| 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:** Only available for Intel Arria 10 and Intel Cyclone 10 GX 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:** Only available for Intel Arria 10 and Intel Cyclone 10 GX 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...
### Table 39. Advanced Fitter Settings (5 of 8)

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimize Design for Metastability</td>
<td>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 <code>report_metastability</code> command to review the synchronizers detected in your design and to produce MTBF estimates.</td>
</tr>
<tr>
<td>Optimize Hold Timing</td>
<td>Directs the Fitter to optimize hold time within a device to meet timing requirements and assignments. The following settings are available:</td>
</tr>
<tr>
<td></td>
<td>•  I/O Paths and Minimum TPD Paths—directs the Fitter to meet the following timing requirements and assignments:</td>
</tr>
<tr>
<td></td>
<td>— t_{H} from I/O pins to registers.</td>
</tr>
<tr>
<td></td>
<td>— Minimum t_{CO} from registers to I/O pins.</td>
</tr>
<tr>
<td></td>
<td>— Minimum t_{PD} from I/O pins or registers to I/O pins or registers.</td>
</tr>
<tr>
<td></td>
<td>•  All Paths—directs the Fitter to meet the following timing requirements and assignments:</td>
</tr>
<tr>
<td></td>
<td>— t_{H} from I/O pins to registers.</td>
</tr>
<tr>
<td></td>
<td>— Minimum t_{CO} from registers to I/O pins.</td>
</tr>
<tr>
<td></td>
<td>— Minimum t_{PD} from I/O pins or registers to I/O pins or registers.</td>
</tr>
<tr>
<td></td>
<td>When you disable the Optimize Timing logic option, the Optimize Hold Timing option is not available.</td>
</tr>
<tr>
<td>Optimize IOC Register Placement for Timing</td>
<td>Specifies whether the Fitter optimizes I/O pin timing by automatically packing registers into I/Os to minimize delays.</td>
</tr>
<tr>
<td></td>
<td>•  Normal—the Fitter opportunistically packs registers into I/Os that should improve I/O timing.</td>
</tr>
<tr>
<td></td>
<td>•  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.</td>
</tr>
<tr>
<td></td>
<td>•  Off—performs no periphery to core optimization.</td>
</tr>
<tr>
<td>Optimize Multi-Corner Timing</td>
<td>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.</td>
</tr>
<tr>
<td>Optimize Timing</td>
<td>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).</td>
</tr>
</tbody>
</table>
### Table 40. Advanced Fitter Settings (6 of 8)

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Periphery to Core Placement and Routing Optimization</td>
<td>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:</td>
</tr>
<tr>
<td></td>
<td>• <strong>Auto</strong>—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.</td>
</tr>
<tr>
<td></td>
<td>• <strong>On</strong>—the Fitter optimizes all transfers between the periphery and core registers, regardless of timing requirements. Do not set this option to <strong>On</strong> globally. Instead, use the Assignment Editor to assign optimization to a targeted set of nodes or entities.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Off</strong>—the Fitter performs no periphery to core optimization.</td>
</tr>
<tr>
<td></td>
<td>Note: Only available for Intel Arria 10 and Intel Cyclone 10 GX devices.</td>
</tr>
<tr>
<td>Physical Placement Effort</td>
<td>Controls how much effort the Fitter spends during advanced physical placement optimization. <strong>High</strong> and <strong>Maximum</strong> effort settings result in additional compile time to further optimize the placement solution.</td>
</tr>
<tr>
<td>Placement Effort Multiplier</td>
<td>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.</td>
</tr>
<tr>
<td>Power Optimization During Fitting</td>
<td>Directs the Fitter to perform optimizations targeted at reducing the total power devices consume. The available settings for power-optimized fitting are:</td>
</tr>
<tr>
<td></td>
<td>• <strong>Off</strong>—performs no power optimizations.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Normal compilation</strong>—performs power optimizations that are unlikely to adversely affect compilation time or design performance.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Extra effort</strong>—performs additional power optimizations that might affect design performance or result in longer compilation time.</td>
</tr>
</tbody>
</table>

### Table 41. Advanced Fitter Settings (7 of 8)

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programmable Power Maximum High-Speed Fraction of Used LAB Tiles</td>
<td>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.</td>
</tr>
<tr>
<td>Programmable Power Technology Optimization</td>
<td>Controls how the Fitter configures tiles to operate in high-speed mode or low-power mode. The following options are available:</td>
</tr>
<tr>
<td></td>
<td>• <strong>Automatic</strong>—specifies that the Fitter minimizes power without sacrificing timing performance.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Minimize Power Only</strong>—specifies that the Fitter sets the maximum number of tiles to operate in low-power mode.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Force All Used Tiles to High Speed</strong>—specifies that the Fitter sets all used tiles to operate in high-speed mode. For designs that meet timing, the behavior of this setting is similar to the <strong>Automatic</strong> 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.</td>
</tr>
</tbody>
</table>
Table 42. Advanced Fitter Settings (8 of 8)

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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. |

**Other Assignments**

```set_global_assignment -name ERROR_ON_INVALID_ENTITY_NAME:```

The software ignores `.qsf` and `.qip` assignments where the entity field is not a name that exists in the design and generates a warning. If you set `ERROR_ON_INVALID_ENTITY_NAME` to ON, the software generates these warnings as errors.

1.13. Design Compilation Revision History

This document has the following revision history.
<table>
<thead>
<tr>
<th>Document Version</th>
<th>Intel Quartus Prime Version</th>
<th>Changes</th>
</tr>
</thead>
</table>
| 2020.11.09       | 20.3                        | • Added new "Integrating Other EDA Tools" topic.  
|                  |                             | • Added new "Generating a VQM Netlist for Other EDA Tools" topic. |
| 2020.09.28       | 20.3                        | • Added references to ECO Compilation Flow.  
|                  |                             | • Removed references to deprecated Early Place Compiler flow. |
| 2020.05.08       | 20.1                        | • Added note about programming file differences between versions to "Compilation Overview" topic. |
| 2020.04.13       | 20.1                        | • Added new "Fast Forward Compile by Hierarchy" topic.  
|                  |                             | • Added new assignment to Fitter Settings Reference.  
|                  |                             | • Updated "Verilog and SystemVerilog Synthesis Support" topic for SystemVerilog 2012 support.  
|                  |                             | • Added programming file generation support for Intel Agilex devices.  
|                  |                             | • Added "Analyzing with the Snapshot Viewer" topic.  
|                  |                             | • Added "Running the Snapshot Viewer" topic.  
|                  |                             | • Added "Analyzing Failing Paths with Snapshot Viewer" topic.  
|                  |                             | • Added "Analyzing Clocking with Snapshot Viewer" topic.  
|                  |                             | • Added "Analyzing High Fan-Out Nets with Snapshot Viewer" topic.  
|                  |                             | • Added "Analyzing Constraints with Snapshot Viewer" topic.  
|                  |                             | • Added "Analyzing Congestion with Snapshot Viewer" topic.  
|                  |                             | • Removed Early Place Flow figure and table.  
|                  |                             | • Removed Heat-Map in Global Signal Visualization Report figure.  
|                  |                             | • Changed sentence in Fast Forward Compilation to The Fitter automatically retimes registers across RAM and DSP blocks from The Fitter does not automatically retime registers across RAM and DSP blocks.  
|                  |                             | • Added more Preservation Level information to Design Partition table. |
| 2019.10.20       | 19.3                        | • Added "Automatic Gated Clock Conversion" topic.  
|                  |                             | • Updated "Fractal Synthesis Optimization" and "Enabling or Disabling Fractal Synthesis" topics for automated fractal synthesis for small multipliers. |
| 2019.09.30       | 19.3                        | • Added support for Intel Agilex devices throughout.  
|                  |                             | • Added "Global Signal Visualization Report" topic.  
|                  |                             | • Added "Global Router Wire Utilization Map" topic.  
|                  |                             | • Added "Fast Preserve Option" topic.  
|                  |                             | • Reordering of some topics to match design flow. |
| 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."  
|                  |                             | • Removed "Enabling Timing-Driven Synthesis" topic. |
| 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." |

continued...
## Document Version | Intel Quartus Prime Version | Changes
--- | --- | ---
2018.10.19 | 18.1 | • Described dependency of Rapid Recompile on Enable Intermediate Fitter Snapshots option.

• 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."

• 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.

2018.09.24 | 18.1 | • 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.

• 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."

• 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 | • 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>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Changes</th>
</tr>
</thead>
</table>
| 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.  
• 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.                                             |
2. Reducing Compilation Time

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

2.1. 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.

2.2. 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.

2.3. 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

2.3.1. Running the ECO Compilation Flow

The Intel Quartus Prime Pro Edition software supports last-minute, targeted design changes (also known as engineering change orders (ECOs)), even after you fully compile the design. ECOs typically occur during the design verification stage. Refer to the Intel Quartus Prime Pro Edition User Guide: Design Optimization for details.
2.3.2. Running Rapid Recompile

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

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
2.3.3. 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.

2.3.4. 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.
When using block-based compilation, you can enable the Fast Preserve option to reduce the logic of preserved partitions to only interface logic during compilation, thereby reducing the time the Compiler requires to synthesize, place, and route the partition. Interface logic is logic at the partition boundary that interfaces with the rest of the design.

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.

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.

4. To enable the Fast Preserve option that simplifies the logic of the preserved partition to only interface logic during compilation, click Assignments ➤ Settings ➤ Compiler Settings ➤ Incremental Compile ➤ Fast Preserve.

**Related Information**

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

### 2.3.5. 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 option is available only for Intel Stratix 10 and Intel Agilex 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.
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:

**Figure 69. Warning for Registers**

To view registers with defined power-up states that the Compiler cannot preserve, refer to the ** Registers with Explicit Power-up Settings** report,
Figure 70. Registers with Explicit Power-up Settings Report

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

2.4. 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.

2.4.1. 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.

2.4.2. 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.

2.5. 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.

2.5.1. 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.

2.6. 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.

2.6.1. 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.

### 2.6.1.1. 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


### 2.6.1.2. 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.

### 2.7. 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.

2.8. 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.

2.9. Reducing Compilation Time Revision History

<table>
<thead>
<tr>
<th>Document Version</th>
<th>Intel Quartus Prime Version</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>2020.09.28</td>
<td>20.3</td>
<td>• Added reference to ECO Compilation flow.</td>
</tr>
<tr>
<td>2019.11.11</td>
<td>19.3</td>
<td>• Added support for Fast Preserve option to &quot;Using Block-Based Compilation&quot; topic.</td>
</tr>
<tr>
<td>2019.09.30</td>
<td>19.3</td>
<td>• Added support for Intel Agilex devices throughout.</td>
</tr>
<tr>
<td>2019.07.02</td>
<td>19.1</td>
<td>Added the Using the No-Register Initialization Flow topic.</td>
</tr>
<tr>
<td>2018.10.19</td>
<td>18.1</td>
<td>• Described dependency of Rapid Recompile on Enable Intermediate Fitter Snapshots option.</td>
</tr>
<tr>
<td>2017.11.06</td>
<td>17.1</td>
<td>• Added topic: Using Block-Based Compilation.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>2017.05.08</td>
<td>17.0.0</td>
<td>• Clarified impact of multiprocessor compilation on fit quality.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Removed reference to deprecated Fitter Effort Logic Option.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Removed section: Preserving Routing with Incremental Compilation.</td>
</tr>
<tr>
<td>2016.10.31</td>
<td>16.1.0</td>
<td>• Implemented Intel rebranding.</td>
</tr>
<tr>
<td>2016.05.02</td>
<td>16.0.0</td>
<td>• Corrected typo in Using Parallel Compilation with Multiple Processors.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Removed information about deprecated physical synthesis options.</td>
</tr>
<tr>
<td>2015.11.02</td>
<td>15.1.0</td>
<td>Changed instances of Quartus II to Intel Quartus Prime.</td>
</tr>
<tr>
<td>2014.12.15</td>
<td>14.1.0</td>
<td>• Updated location of Fitter Settings, Analysis &amp; Synthesis Settings, and Physical Synthesis Optimizations to Compiler Settings.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added information about Rapid Recompile feature.</td>
</tr>
</tbody>
</table>

continued...
<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>2014.08.18</td>
<td>14.0a10.0</td>
<td>Added restriction about smart compilation in Arria 10 devices.</td>
</tr>
<tr>
<td>June 2014</td>
<td>14.0.0</td>
<td>Updated format.</td>
</tr>
<tr>
<td>June 2012</td>
<td>12.0.0</td>
<td>Removed survey link.</td>
</tr>
<tr>
<td>November 2011</td>
<td>11.0.1</td>
<td>Template update.</td>
</tr>
<tr>
<td>May 2011</td>
<td>11.0.0</td>
<td>• Updated “Using Parallel Compilation with Multiple Processors”.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Updated “Identifying Routing Congestion in the Chip Planner”.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• General editorial changes throughout the chapter.</td>
</tr>
<tr>
<td>December 2010</td>
<td>10.1.0</td>
<td>• Template update.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added details about peak and average interconnect usage.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added new section &quot;Reducing Static Timing Analysis Time&quot;.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Minor changes throughout chapter.</td>
</tr>
<tr>
<td>July 2010</td>
<td>10.0.0</td>
<td>Initial release.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Intel Quartus Prime Version</th>
<th>User Guide</th>
</tr>
</thead>
<tbody>
<tr>
<td>18.1</td>
<td>Intel Quartus Prime Pro Edition User Guide Design Compilation</td>
</tr>
<tr>
<td>18.0</td>
<td>Compiler User Guide Intel Quartus Prime Pro Edition</td>
</tr>
</tbody>
</table>
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.

  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.

  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.

  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).

  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*.

• 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, system debugging toolkits, In-System Memory Content Editor, and In-System Sources and Probes Editor.

  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.

  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.

  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.