This chapter describes techniques to improve timing performance when designing for Altera® devices.

The application techniques vary between designs. Applying each technique does not always improve results. Settings and options in the Quartus® II software have default values that provide the best trade-off between compilation time, resource utilization, and timing performance. You can adjust these settings to determine whether other settings provide better results for your design.

**Initial Compilation: Optional Fitter Settings**

The Fitter offers many optional settings; however, this section discusses important optional timing-optimization related Fitter settings only, which are the Optimize Hold Timing, Optimize Multi-Corner Timing, and Fitter Aggressive Routability Optimization settings.

For scripting and device family support information of the Optimize Hold Timing and Optimize Multi-Corner Timing settings, refer to the Fitter Settings Page (Settings Dialog Box) in Quartus II Help.

The settings required to optimize different designs could be different. The group of settings that work best for one design may not produce the best result for another design.

**Optimize Hold Timing**

The Optimize Hold Timing option directs the Quartus II software to optimize minimum delay timing constraints. By default, the Quartus II software optimizes hold timing for all paths for designs using devices newer than Arria® GX, Stratix® III, and Cyclone® III. By default, the Quartus II software optimizes hold timing only for I/O paths and minimum tPD paths for older devices.

When you turn on Optimize Hold Timing, the Quartus II software adds delay to paths to ensure that your design meets the minimum delay requirements. In the Fitter Settings pane, if you select I/O Paths and Minimum TPD Paths (the default choice for older devices such as Cyclone II and Stratix II devices if you turn on Optimize Hold Timing), the Fitter works to meet the following criteria:

- Hold times (t_H) from the device input pins to the registers
- Minimum delays from I/O pins to I/O registers or from I/O registers to I/O pins
- Minimum clock-to-out time (t_CO) from registers to output pins
If you select **All Paths**, the Fitter also works to meet hold requirements from registers to registers, as highlighted in blue in **Figure 12–1**, in which a derived clock generated with logic causes a hold time problem on another register.

**Figure 12–1. Optimize Hold Timing Option Fixing an Internal Hold Time Violation**

However, if your design still has internal hold time violations between registers, correct the problems by manually adding some delays by instantiating LCELL primitives, or by making changes to your design, such as using a clock enable signal instead of a derived or gated clock.

For design practices that helps eliminate internal hold time violations, refer to the *Recommended Design Practices* chapter of the *Quartus II Handbook*.

**Optimize Multi-Corner Timing**

Due to process variations and changes in operating conditions, delays on some paths can be significantly smaller than those in the slow corner timing model. This can result in hold time violations on those paths, and in rare cases, additional setup time violations.

Also, because of the small process geometries of the Arria GX, Cyclone III, Stratix III, and newer device families, the slowest circuit performance of designs targeting these devices does not necessarily occur at the highest operating temperature. The temperature at which the circuit is slowest depends on the selected device, the design, and the compilation results. Therefore, the Quartus II software provides the Arria GX, Cyclone III series, Stratix III, and newer device families with three different timing corners—Slow 85°C corner, Slow 0°C corner, and Fast 0°C corner. For other device families, two timing corners are available—Fast 0°C and Slow 85°C corner.

The **Optimize multi-corner timing** option directs the Fitter to consider all corner timing delays, including both fast-corner timing and slow-corner timing, during optimization to meet timing requirements at all process corners and operating conditions. By default, this option is on, and the Fitter optimizes designs considering multi-corner delays in addition to slow-corner delays, for example, from the fast-corner timing model, which is based on the fastest manufactured device, operating under high-voltage conditions.

The **Optimize multi-corner timing** option helps to create a design implementation that is more robust across process, temperature, and voltage variations. Turning on this option increases compilation time by approximately 10%.
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).

**Fitter Aggressive Routability Optimization**

The **Fitter Aggressive Routability Optimizations** logic option allows you to specify 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.

This option is useful if routing resources are resulting in no-fit errors, and you want to reduce routing wire use.

Table 12–1 lists the settings for the **Fitter Aggressive Routability Optimizations** logic option.

### Table 12–1. Fitter Aggressive Routability Optimizations Logic Option Settings

<table>
<thead>
<tr>
<th>Settings</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Always</td>
<td>The Fitter always performs aggressive routability optimizations. If you set the Fitter Aggressive Routability Optimizations logic option to Always, reducing wire utilization may affect the performance of your design.</td>
</tr>
<tr>
<td>Never</td>
<td>The Fitter never performs aggressive routability optimizations. If improving timing is more important than reducing wire usage, then set this option to Automatically or Never.</td>
</tr>
<tr>
<td>Automatically</td>
<td>The Fitter performs aggressive routability optimizations automatically, based on the routability and timing requirements of the design. If improving timing is more important than reducing wire usage, then set this option to Automatically or Never.</td>
</tr>
</tbody>
</table>

**Design Analysis**

The initial compilation establishes whether the design achieves a successful fit and meets the specified timing requirements. This section describes how to analyze your design results in the Quartus II software.

**Ignored Timing Constraints**

The Quartus II software ignores illegal, obsolete, and conflicting constraints.

You can view a list of ignored constraints by clicking **Report Ignored Constraints** in the Reports menu in the TimeQuest GUI or by typing the following command to generate a list of ignored timing constraints:

```
report_sdc -ignored -panel_name "Ignored Constraints"
```

You should analyze any constraints that the Quartus II software ignores. If necessary, correct the constraints and recompile your design before proceeding with design optimization.

For more information about the **report_sdc** command and its options, refer to the **Quartus II TimeQuest Timing Analyzer** chapter of the **Quartus II Handbook**.
You can view a list of ignored assignment in the **Ignored Assignment Report** generated by the Fitter. For more information, refer to the **Fitter Summary Reports** in the Quartus II Help.

### I/O Timing (Including $t_{PD}$)

TimeQuest analyzer supports the Synopsys Design Constraints (SDC) format for constraining your design. When using the TimeQuest analyzer for timing analysis, use the `set_input_delay` constraint to specify the data arrival time at an input port with respect to a given clock. For output ports, use the `set_output_delay` command to specify the data arrival time at an output port's receiver with respect to a given clock. You can use the `report_timing` Tcl command to generate the I/O timing reports.

The I/O paths that do not meet the required timing performance are reported as having negative slack and are highlighted in red in the TimeQuest analyzer **Report** pane. In cases where you do not apply an explicit I/O timing constraint to an I/O pin, the Quartus II timing analysis software still reports the **Actual number**, which is the timing number that must be met for that timing parameter when the device runs in your system.

For more information about how timing numbers are calculated, refer to **The Quartus II TimeQuest Timing Analyzer** chapter of the **Quartus II Handbook**.

### Register-to-Register Timing

This section contains the following sections:

- “Timing Analysis with the TimeQuest Timing Analyzer”
- “Tips for Analyzing Failing Paths” on page 12–6
- “Tips for Analyzing Failing Clock Paths that Cross Clock Domains” on page 12–6
- “Tips for Analyzing Paths from/to the Source and Destination of Critical Path” on page 12–7
- “Tips for Locating Multiple Paths to the Chip Planner” on page 12–8
- “Tips for Creating a .tcl Script to Monitor Critical Paths Across Compiles” on page 12–9
- “Global Routing Resources” on page 12–9

### Timing Analysis with the TimeQuest Timing Analyzer

Analyze all valid register-to-register paths by using the appropriate constraints in the TimeQuest analyzer. To view all timing summaries, run the **Report All Summaries** command by double-clicking **Report All Summaries** in the Tasks pane in the TimeQuest analyzer.

If any clock domains have failing paths (highlighted in red in the Report panel), right-click the Clock Name listed in the Clocks Summary panel and go to **Report Timing** to get more details. Your design meets timing requirements when you do not have negative slack on any register-to-register path on any of the clock domains.
When timing requirements are not met, a report on the failure paths (highlighted in red) can uncover more detail.

When you select a path listed in the TimeQuest **Report Timing** pane, the tabs in the corresponding path detail pane show a path summary of source and destination registers and their timing, statistics about the path delay, detailed information about the complete data path with all nodes in the path, and the waveforms of the relevant signals. The **Extra Fitter Information** tab will show a Graphical Data Path of where the offending path lies on the physical device. This can reveal whether the timing failure may be distance related, due to the source and destination node being too close or too far. The Chip Planner can also be used to investigate the physical layout of a failing path in more detail. To locate a selected path in the Chip Planner, right-click a node, point to **Locate**, and select **Locate in Chip Planner**. The Chip Planner appears with the path highlighted. Use this to show fanout, fanin, routing congestion, and region assignments information, and to determine whether those factors might be contributing to the timing critical path. Additionally, if you know that a path is not a valid path, you can set it to be a false path using the shortcut menu.

The **Data Path** tab can also be useful for determining contributions to timing critical paths. The **Data Path** tab shows details of the paths that the clock and data took to get from source to destination nodes, and the time it took on an incremental and cumulative basis. It also provides information about the routing types and elements used, and their locations.

To view the path details of any selected path, click the **Data Path** tab in the path details pane. The **Data Path** tab displays the details of the Data Arrival Path, as well as the Data Required Path.

The **Waveform** tab will show the slack relationship between arrival data and required data. This could be useful for determining how close or far off the path is from meeting timing.

For more information about how timing analysis results are calculated, refer to *The Quartus II TimeQuest Timing Analyzer* chapter of the *Quartus II Handbook*.

To aid in timing debug, the RTL Viewer or Technology Map Viewer allow you to see schematic representations of your design. These viewers allow you to view a gate-level or technology-mapped representation of your design netlist. By providing a view of the path from source and destination nodes, the viewers can help identify areas in a design that may benefit from reducing the number of logic levels between the nodes. To locate a timing path in one of the viewers, right-click a path in the report, point to **Locate**, and click **Locate in RTL Viewer** or **Locate in Technology Map Viewer**.

For more information about netlist viewers, refer to the *Analyzing Designs with Quartus II Netlist Viewers* chapter of the *Quartus II Handbook*. 
**Tips for Analyzing Failing Paths**

When you are analyzing failing paths, examine the reports and waveforms to determine if the correct constraints are being applied, and add timing exceptions as appropriate. A multicycle constraint relaxes setup or hold relationships by the specified number of clock cycles. A false path constraint specifies paths that can be ignored during timing analysis. Both constraints allow the Fitter to work harder on affected paths.

Focus on improving the paths that show the worst slack. The Fitter works hardest on paths with the worst slack. If you fix these paths, the Fitter might be able to improve the other failing timing paths in the design. For more information, refer to “Design Evaluation for Timing Closure” on page 12–27.

Check for particular nodes that appear in many failing paths. These nodes will appear in a timing report panel at the top of the list, along with their minimum slacks. Look for paths that have common source registers, destination registers, or common intermediate combinational nodes. In some cases, the registers might not be identical, but are part of the same bus.

In the timing analysis report panels, clicking on the From or To column headings can help to sort the paths by the source or destination registers. Clicking first on From, then on To, uses the registers in the To column as the primary sort and the registers in the From column as the secondary sort. If you see common nodes, these nodes indicate areas of your design that might be improved through source code changes or Quartus II optimization settings. Constraining the placement for just one of the paths might decrease the timing performance for other paths by moving the common node further away in the device.

**Tips for Analyzing Failing Clock Paths that Cross Clock Domains**

When analyzing clock path failures, check whether these paths cross two clock domains. This is the case if the From Clock and To Clock in the timing analysis report are different (Figure 12–2).

There can also be paths that involve a different clock in the middle of the path, even if the source and destination register clock are the same.

When you run Report Timing on your design, the report shows the launch clock and latch clock for each failing path. Check whether these failing paths between these clock domains should be analyzed synchronously. If the failing paths are not to be analyzed synchronously, they must be set as false paths. Also check the relationship between the launch clock and latch clock to make sure it is realistic and what you expect from your knowledge of the design. For example, the path can start at a rising edge and end at a falling edge, which reduces the setup relationship by one half clock cycle.
Review the clock skew reported in the Timing Report. A large skew may indicate a problem in your design, such as a gated clock or a problem in the physical layout (for example, a clock using local routing instead of dedicated clock routing). When you have made sure the paths are analyzed synchronously and that there is no large skew on the path, and that the constraints are correct, you can analyze the data path. These steps help you fine tune your constraints for paths across clock domains to ensure you get an accurate timing report.

Check if the PLL phase shift is reducing the setup requirement. You might be able to adjust this using PLL parameters and settings.

Paths that cross clock domains are generally protected with synchronization logic (for example, FIFOs or double-data synchronization registers) to allow asynchronous interaction between the two clock domains. In such cases, you can ignore the timing paths between registers in the two clock domains while running timing analysis, even if the clocks are related.

The Fitter attempts to optimize all failing timing paths. If there are paths that can be ignored for optimization and timing analysis, but the paths do not have constraints that instruct the Fitter to ignore them, the Fitter tries to optimize those paths as well. In some cases, optimizing unnecessary paths can prevent the Fitter from meeting the timing requirements on timing paths that are critical to the design. It is beneficial to specify all paths that can be ignored by setting false path constraints on them, so that the Fitter can put more effort into the paths that must meet their timing requirements instead of optimizing paths that can be ignored.

For more details about how to ignore timing paths that cross clock domains, refer to The Quartus II TimeQuest Timing Analyzer chapter of the Quartus II Handbook.

### Tips for Analyzing Paths from/to the Source and Destination of Critical Path

When analyzing the failing paths in a design, it is often helpful to get a fuller picture of the many interactions the fitter may be working on around the paths. To understand what may be pulling on a critical path, the following report_timing command can be useful.

In the project directory, run the Tcl command shown in Example 12–1 in a .tcl file to analyze the nodes in a critical path.

**Example 12–1. report_timing Command**

```tcl
set wrst_src <insert_source_of_worst_path_here>
set wrst_dst <insert_destination_of_worst_path_here>
report_timing -setup -npaths 50 -detail path_only -from $wrst_src -panel_name "Worst Path|wrst_src -> *"
report_timing -setup -npaths 50 -detail path_only -to $wrst_dst -panel_name "Worst Path|* -> wrst_dst"
report_timing -setup -npaths 50 -detail path_only -to $wrst_src -panel_name "Worst Path|* -> wrst_src"
report_timing -setup -npaths 50 -detail path_only -from $wrst_dst -panel_name "Worst Path|wrst_dst -> *"
```

Copy the node names from the From Node and To Node columns of the worst path into the first two variables, and then in the TimeQuest timing analyzer, in the Script menu, source the .tcl script.
In the resulting timing panel, timing failed paths (highlighted in red) can be located in the Chip Planner, where information such as distance between the nodes and large fanouts can be viewed.

Figure 12–3 shows a simplified example of what these reports analyzed.

**Figure 12–3. Timing Report**

The critical path of the design is in red. The script analyzes the path between the worst source and destination registers. The first `report_timing` command analyzes other path that the source is driving, as shown in green. The second `report_timing` command analyzes the critical path and other path going to the destination, shown in yellow. These commands report everything inside these two endpoints that are pulling them in different directions. The last two `report_timing` commands show everything outside of the endpoints pulling them in other directions. If any of these reports have slacks near the critical path, then the Fitter is balancing these paths with the critical path, trying to achieve the best slack. Figure 12–3 is quite simple compared to the critical path in most designs, but it is easy to see how this can get very complicated quickly.

**Tips for Locating Multiple Paths to the Chip Planner**

The Chip Planner can be used as a visual aid in locating timing critical paths. To view these paths from timing reports, do the following:

1. Run `report_timing` to show multiple paths. *(Example 12–1)*
2. Select multiple rows of timing report.
3. Right-click, select **Locate Path**, and then click **Chip Planner**.
4. The **Locate History** window in the Chip Planner displays the selected paths and the worst path.
5. Double-click **Locate Paths** to show all paths at once, or select individual paths to view the path in the Chip Planner.
This will show whether timing failures may be due to large distances between the nodes or large fanouts.

**Tips for Creating a .tcl Script to Monitor Critical Paths Across Compiles**

Many designs have the same critical paths show up after each compile, but some suffer from having critical paths bounce around between different hierarchies, changing with each compile.

This could happen in high speed designs where many register to register paths have very little slack. Different placements can then result in timing failures in the marginal paths. In designs like this, create a `TQ_critical_paths.tcl` script in the project directory.

For a given compile, view the critical paths and then write a generic `report_timing` command to capture those paths. For example, if several paths fail in a low-level hierarchy, you can add the following command as shown in Example 12–2:

**Example 12–2. report_timing Command**

```tcl
report_timing -setup -npaths 50 -detail path_only -to "main_system:main_system_inst|app_cpu:cpu:*" -panel_name "Critical Paths||s: * -> app_cpu"
```

If there is a specific path, such as a bit of a state-machine going to other `*count_sync*` registers, you can add a command as shown in Example 12–3:

**Example 12–3. report_timing Command**

```tcl
report_timing -setup -npaths 50 -detail path_only -from "main_system:main_system_inst|egress_count_sm:egress_inst|update" -to "*count_sync*" -panel_name "Critical Paths||s: egress_sm|update -> count_sync"
```

This file can be sourced in the TimeQuest timing analyzer after every compilation, and new `report_timing` commands can be added as new critical paths appear. This helps you monitor paths that consistently fail and paths that are only marginal, so you can prioritize effectively.

**Global Routing Resources**

Global routing resources are designed to distribute high fan-out, low-skew signals (such as clocks) without consuming regular routing resources. Depending on the device, these resources can span the entire chip, or some smaller portion, such as a quadrant. The Quartus II software attempts to assign signals to global routing resources automatically, but you might be able to make more suitable assignments manually.

For details about the number and types of global routing resources available, refer to the relevant device handbook.

Check the global signal utilization in your design to ensure that the appropriate signals have been placed on the global routing resources. In the Compilation Report, open the Fitter report and click **Resource Section**. Analyze the Global & Other Fast Signals and Non-Global High Fan-out Signals reports to determine whether any changes are required.
You might be able to reduce skew for high fan-out signals by placing them on global routing resources. Conversely, you can reduce the insertion delay of low fan-out signals by removing them from global routing resources. Doing so can improve clock enable timing and control signal recovery/removal timing, but increases clock skew. Use the Global Signal setting in the Assignment Editor to control global routing resources.

**Optimizing Timing (LUT-Based Devices)**

This section contains guidelines that might help you if your design does not meet its timing requirements.

**Debugging Timing Failures in the TimeQuest Analyzer**

A Report Timing Closure Recommendations task is available in the Custom Reports section of the Tasks pane of the TimeQuest analyzer. Use this report to get more information and help on the failing paths in your design. This feature is available for the Arria II, Cyclone III, Stratix III, and newer devices.

When you run the Report Timing Closure Recommendations task, you get specific recommendations about failing paths in your design and changes that you can make to potentially fix the failing paths.


From the Report Timing Closure Recommendations dialog box, you can select paths based on the clock domain, filter by nodes on path, and choose the number of paths to analyze.

After running the Report Timing Closure Recommendations task in the TimeQuest analyzer, examine the reports in the Report Timing Closure Recommendations folder in the Report pane of the TimeQuest analyzer GUI. Each recommendation has star symbols (*) associated with it. Recommendations with more stars are more likely to help you close timing on your design.

The reports give you the most probable causes of failure for each path being analyzed. The reports are organized into sections, depending on the type of issues found in the design, such as large clock skew, restricted optimizations, unbalanced logic, skipped optimizations, coding style that has too many levels of logic between registers, or region or partition constraints specific to your project.

You will see recommendations that may help you fix the failing paths. For detailed analysis of the critical paths, run the report_timing command on specified paths. In the Extra Fitter Information tab of the Path report panel, you will also see detailed Fitter-related information that may help you visualize the issue and take the appropriate action if your constraints cause a specific placement.

Timing Optimization Advisor

While the TimeQuest Report Timing Closure Recommendations task gives specific recommendations to fix failing paths, the Timing Optimization Advisor gives more general recommendations to improve timing performance for a design.

The Timing Optimization Advisor guides you in making settings that optimize your design to meet your timing requirements. To run the Timing Optimization Advisor, on the Tools menu, point to Advisors and click Timing Optimization Advisor. This advisor describes many of the suggestions made in this section.

When you open the Timing Optimization Advisor after compilation, you can find recommendations to improve the timing performance of your design. Some of the recommendations in these advisors can contradict each other. Altera recommends evaluating these options and choosing the settings that best suit the given requirements.

Figure 12–4 shows the Timing Optimization Advisor after compiling a design that meets its frequency requirements, but requires setting changes to improve the timing.

Figure 12–4. Timing Optimization Advisor

When you expand one of the categories in the Timing Optimization Advisor, such as Maximum Frequency (fmax) or I/O Timing (tsu, tco, tpd), the recommendations are divided into stages. The stages show the order in which to apply the recommended settings. The first stage contains the options that are easiest to change, make the least drastic changes to your design optimization, and have the least effect on compilation time. Icons indicate whether each recommended setting has been made in the current project. In Figure 12–4, the checkmark icons in the list of recommendations for Stage 1 indicate recommendations that are already implemented. The warning icons indicate
recommendations that are not followed for this compilation. The information icons indicate general suggestions. For these entries, the advisor does not report whether these recommendations were followed, but instead explains how you can achieve better performance. For a legend that provides more information for each icon, refer to the “How to use” page in the Timing Optimization Advisor.

There is a link from each recommendation to the appropriate location in the Quartus II GUI where you can change the settings. For example, consider the Synthesis Netlist Optimizations page of the Settings dialog box or the Global Signals category in the Assignment Editor. This approach provides the most control over which settings are made and helps you learn about the settings in the software. In some cases, you can also use the Correct the Settings button to automatically make the suggested change to global settings.

For some entries in the Timing Optimization Advisor, a button appears that allows you to further analyze your design and gives you more information. The advisor provides a table with the clocks in the design and indicates whether they have been assigned a timing constraint.

### I/O Timing Optimization

This stage of design optimization focuses on I/O timing. Ensure that you have made the appropriate assignments described in the “Initial Compilation: Required Settings” section in the Design Optimization Overview chapter of the Quartus II Handbook. You must also ensure that resource utilization is satisfactory before proceeding with I/O timing optimization. The suggestions provided in this section are applicable to all Altera FPGA families and to the MAX II family of CPLDs.

Because changes to the I/O paths affect the internal register-to-register timing, complete this stage before proceeding to the register-to-register timing optimization stage as described in “Register-to-Register Timing Optimization Techniques (LUT-Based Devices)” on page 12–17.

The options presented in this section address how to improve I/O timing, including the setup delay (\(t_{SU}\)), hold time (\(t_{H}\)), and clock-to-output (\(t_{CO}\)) parameters.

### Improving Setup and Clock-to-Output Times Summary

Table 12–2 lists the recommended order in which to use techniques to reduce \(t_{SU}\) and \(t_{CO}\) times. Checkmarks indicate which timing parameters are affected by each technique. Reducing \(t_{SU}\) times increases hold (\(t_{H}\)) times.

<table>
<thead>
<tr>
<th>Table 12–2. Improving Setup and Clock-to-Output Times ((^f)) (Part 1 of 2)</th>
<th>(t_{SU})</th>
<th>(t_{CO})</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ensure that the appropriate constraints are set for the failing I/Os (refer to the “Initial Compilation: Required Settings” section in the Design Optimization Overview chapter of the Quartus II Handbook.)</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Use timing-driven compilation for I/O (page 12–13)</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Use fast input register (page 12–14)</td>
<td>✓</td>
<td>—</td>
</tr>
<tr>
<td>Use fast output register, fast output enable register, and fast OCT register (page 12–14)</td>
<td>—</td>
<td>✓</td>
</tr>
<tr>
<td>Decrease the value of Input Delay from Pin to Input Register or set Decrease Input Delay to Input Register = ON</td>
<td>✓</td>
<td>—</td>
</tr>
</tbody>
</table>
Timing-Driven Compilation

This option moves registers into I/O elements if required to meet $t_{SU}$ or $t_{CO}$ assignments, duplicating the register if necessary (as in the case in which a register fans out to multiple output locations). This option is turned on by default and is a global setting. The option does not apply to MAX II series devices because they do not contain I/O registers.

The Optimize IOC Register Placement for Timing option affects only pins that have a $t_{SU}$ or $t_{CO}$ requirement. Using the I/O register is possible only if the register directly feeds a pin or is fed directly by a pin. This setting does not affect registers with any of the following characteristics:

- Have combinational logic between the register and the pin
- Are part of a carry or cascade chain
- Have an overriding location assignment
- Use the asynchronous load port and the value is not 1 (in device families where the port is available)

Registers with the characteristics listed are optimized using the regular Quartus II Fitter optimizations.

For more information, refer to Optimize IOC Register Placement for Timing logic option in Quartus II Help.

Fast Input, Output, and Output Enable Registers

Normally, with correct timing assignments, the Fitter already places the I/O registers in the correct I/O cell or in the core, to meet the performance requirement. However, you can place individual registers in I/O cells manually by making fast I/O assignments with the Assignment Editor.
For more information about the **Fast Input Register** option, **Fast Output Register** option, **Fast Output Enable Register** option, and **Fast OCT (on-chip termination) Register** option, refer to Quartus II Help.

In MAX II series devices, which have no I/O registers, these assignments lock the register into the LAB adjacent to the I/O pin if there is a pin location assignment for that I/O pin.

If the fast I/O setting is on, the register is always placed in the I/O element. If the fast I/O setting is off, the register is never placed in the I/O element. This is true even if the **Optimize IOC Register Placement for Timing** option is turned on. If there is no fast I/O assignment, the Quartus II software determines whether to place registers in I/O elements if the **Optimize IOC Register Placement for Timing** option is turned on.

You can also use the four fast I/O options (**Fast Input Register**, **Fast Output Register**, **Fast Output Enable Register**, and **Fast OCT Register**) to override the location of a register that is in a LogicLock region and force it into an I/O cell. If you apply this assignment to a register that feeds multiple pins, the register is duplicated and placed in all relevant I/O elements. In MAX II series devices, the register is duplicated and placed in each distinct LAB location that is next to an I/O pin with a pin location assignment.

### Programmable Delays

You can use various programmable delay options to minimize the $t_{SU}$ and $t_{CO}$ times. For Arria, Cyclone, MAX II, MAX V, and Stratix series devices, the Quartus II software automatically adjusts the applicable programmable delays to help meet timing requirements. Programmable delays are advanced options to use only after you compile a project, check the I/O timing, and determine that the timing is unsatisfactory. For detailed information about the effect of these options, refer to the device family handbook or data sheet.

After you have made a programmable delay assignment and compiled the design, you can view the implemented delay values for every delay chain for every I/O pin in the **Delay Chain Summary** section of the Compilation Report.

You can assign programmable delay options to supported nodes with the Assignment Editor. You can also view and modify the delay chain setting for the target device with the Chip Planner and Resource Property Editor. When you use the Resource Property Editor to make changes after performing a full compilation, recompiling the entire design is not necessary; you can save changes directly to the netlist. Because these changes are made directly to the netlist, the changes are not made again automatically when you recompile the design. The change management features allow you to reapply the changes on subsequent compilations.

Although the programmable delays in newer devices are user-controllable, Altera recommends their use for advanced users only. However, the Quartus II software might use the programmable delays internally during the Fitter phase.

For more information about Stratix III programmable delays, refer to the **Stratix III Device Handbook** and **AN 474: Implementing Stratix III Programmable I/O Delay Settings in the Quartus II Software**. For more information about using the Chip Planner and Resource Property Editor, refer to the **Engineering Change Management with the Chip Planner** chapter of the **Quartus II Handbook**.
For details about the programmable delay logic options available for Altera devices, refer to the following Quartus II Help topics:

- Decrease Input Delay to Input Register logic option
- Input Delay from Pin to Input Register logic option
- Decrease Input Delay to Internal Cells logic option
- Input Delay from Pin to Internal Cells logic option
- Decrease Input Delay to Output Register logic option
- Increase Delay to Output Enable Pin logic option
- Output Enable Pin Delay logic option
- Increase Delay to Output Pin logic option
- Delay from Output Register to Output Pin logic option
- Increase Input Clock Enable Delay logic option
- Input Delay from Dual-Purpose Clock Pin to Fan-Out Destinations logic option
- Increase Output Clock Enable Delay logic option
- Increase Output Enable Clock Enable Delay logic option
- Increase t<sub>zx</sub> Delay to Output Pin logic option

**Use PLLs to Shift Clock Edges**

Using a PLL typically improves I/O timing automatically. If the timing requirements are still not met, most devices allow the PLL output to be phase shifted to change the I/O timing. Shifting the clock backwards gives a better t<sub>H</sub> at the expense of t<sub>SU</sub>, while shifting it forward gives a better t<sub>SU</sub> at the expense of t<sub>H</sub> (refer to Figure 12–5). You can use this technique only in devices that offer PLLs with the phase shift option.

*Figure 12–5. Shift Clock Edges Forward to Improve t<sub>SU</sub> at the Expense of t<sub>H</sub>*

You can achieve the same type of effect in certain devices by using the programmable delay called Input Delay from Dual Purpose Clock Pin to Fan-Out Destinations.

For more information, refer to Input Delay from Dual-Purpose Clock Pin to Fan-Out Destinations logic option in Quartus II Help.

**Use Fast Regional Clock Networks and Regional Clocks Networks**

Altera devices have a variety of hierarchical clock structures. These include dedicated global clock networks, regional clock networks, fast regional clock networks, and periphery clock networks. The available resources differ between the various Altera device families.
For the number of clocking resources available in your target device, refer to the appropriate device handbook.

In general, fast regional clocks have less delay to I/O elements than regional and global clocks, and are used for high fan-out control signals. Regional clocks provide the lowest clock delay and skew for logic contained in a single quadrant. Placing clocks on these low-skew and low-delay clock nets provides better $t_{CO}$ performance.

**Spine Clock Limitations**

Global clock networks, regional clock networks, and periphery clock networks have an additional level of clock hierarchy known as spine clocks. Spine clocks drive the final row and column clocks to their registers; thus, the clock to every register in the chip is reached through spine clocks. Spine clocks are not directly user controllable.

If your project has high clock routing demands, due to limitations in the Quartus II software, you may see spine clock errors. These errors are often seen with designs using multiple memory interfaces and high-speed serial interface (HSSI) channels (especially PMA Direct mode).

To reduce these spine clock errors, you can constrain your design to better use your regional clock resources using the following techniques:

- If your design does not use LogicLock regions, or if the LogicLock regions are not aligned to your clock region boundaries, create additional LogicLock regions and further constrain your logic.

  - Register packing, a Fitter optimization option, may ignore LogicLock regions. If this occurs, disable register packing for specific instances through the Quartus II Assignment Editor.

- Some periphery features may ignore LogicLock region assignments. When this happens, the global promotion process may not function properly. To ensure that the global promotion process uses the correct locations, assign specific pins to the I/Os using these periphery features.

- By default, some IP MegaCore functions apply a global signal assignment with a value of dual-regional clock. If you constrain your logic to a regional clock region and set the global signal assignment to Regional instead of Dual-Regional, you can reduce clock resource contention.

**Change How Hold Times are Optimized for MAX II Devices**

For MAX II devices, you can use the Guarantee I/O Paths Have Zero Hold Time at Fast Corner option to control how hold time is optimized by the Quartus II software.

For details, refer to Guarantee I/O Paths Have Zero Hold Time at Fast Corner logic option in Quartus II Help.
Register-to-Register Timing Optimization Techniques (LUT-Based Devices)

The next stage of design optimization is to improve register-to-register ($f_{\text{MAX}}$) timing. The following sections provide available options if the performance requirements are not achieved after compilation.

Coding style affects the performance of your design to a greater extent than other changes in settings. Always evaluate your code and make sure to use synchronous design practices.

For more details about synchronous design practices and coding styles, refer to the Recommended Design Practices chapter of the Quartus II Handbook.

When using the TimeQuest analyzer, register-to-register timing optimization is the same as maximizing the slack on the clock domains in your design. You can use the techniques described in this section to improve the slack on different timing paths in your design.

Before optimizing your design, understand the structure of your design as well as the type of logic affected by each optimization. An optimization can decrease performance if the optimization does not benefit your logic structure.

Optimize Source Code

In many cases, optimizing the design’s source code can have a very significant effect on your design performance. In fact, optimizing your source code is typically the most effective technique for improving the quality of your results and is often a better choice than using LogicLock or location assignments.

Be aware of the number of logic levels needed to implement your logic while you are coding. Too many levels of logic between registers could result in critical paths failing timing. Try restructuring the design to use pipelining or more efficient coding techniques. Also, try limiting high fan-out signals in the source code. When possible, duplicate and pipeline control signals. Make sure the duplicate registers are protected by a preserve attribute, to avoid merging during synthesis.

If the critical path in your design involves memory or DSP functions, check whether you have code blocks in your design that describe memory or functions that are not being inferred and placed in dedicated logic. You might be able to modify your source code to cause these functions to be placed into high-performance dedicated memory or resources in the target device. When using RAM/DSP blocks, enable the optional input and output registers.

Ensure that your state machines are recognized as state machine logic and optimized appropriately in your synthesis tool. State machines that are recognized are generally optimized better than if the synthesis tool treats them as generic logic. In the Quartus II software, you can check the State Machine report under Analysis & Synthesis in the Compilation Report. This report provides details, including state encoding for each state machine that was recognized during compilation. If your state machine is not recognized, you might have to change your source code to enable it to be recognized.
For coding style guidelines including examples of HDL code for inferring memory, functions, guidelines, and sample HDL code for state machines, refer to the Recommended HDL Coding Styles chapter of the Quartus II Handbook.

For additional HDL coding examples, refer to AN 584: Timing Closure Methodology for Advanced FPGA Designs.

Improving Register-to-Register Timing Summary

The choice of options and settings to improve the timing margin (slack) or to improve register-to-register timing depends on the failing paths in the design. To achieve the results that best approximate your performance requirements, apply the following techniques and compile the design after each step:

1. Ensure that your timing assignments are complete and correct. For details, refer to the “Initial Compilation: Required Settings” section in the Design Optimization Overview chapter of the Quartus II Handbook.

2. Ensure that you have reviewed all warning messages from your initial compilation and check for ignored timing assignments.

   For details and to fix any of these problems before proceeding with optimization, refer to the Design Optimization Overview chapter of the Quartus II Handbook.

3. Apply netlist synthesis optimization options.

4. To optimize for speed, apply the following synthesis options:
   - “Optimize Synthesis for Speed, Not Area” on page 12–20
   - “Flatten the Hierarchy During Synthesis” on page 12–21
   - “Set the Synthesis Effort to High” on page 12–21
   - “Change State Machine Encoding” on page 12–22
   - “Prevent Shift Register Inference” on page 12–23
   - “Use Other Synthesis Options Available in Your Synthesis Tool” on page 12–23

5. To optimize for performance using physical synthesis, apply the following options:
   - Perform physical synthesis for combinational logic
   - Perform automatic asynchronous signal pipelining
   - Perform register duplication
   - Perform register retiming
   - Perform logic to memory mapping

   For more information about physical synthesis optimization, refer to “Physical Synthesis Optimizations” on page 12–19.

6. Try different Fitter seeds (page 12–23). If there are very few paths that are failing by small negative slack, then you can try with a different seed to see if there is a fit that meets constraints in the Fitter seed noise.
Omit this step if a large number of critical paths are failing or if the paths are failing badly.

7. To control placement, make LogicLock assignments (page 12–24).

8. Make design source code modifications to fix areas of the design that are still failing timing requirements by significant amounts (page 12–17).

9. Make location assignments, or as a last resort, perform manual placement by back-annotating the design (page 12–26).

You can use the Design Space Explorer (DSE) to automate the process of running several different compilations with different settings.

For more information, refer to About Design Space Explorer in Quartus II Help.

If these techniques do not achieve performance requirements, additional design source code modifications might be required (page 12–17).

Physical Synthesis Optimizations

The Quartus II software offers physical synthesis optimizations that can help improve the performance of many designs regardless of the synthesis tool used. Physical synthesis optimizations can be applied both during synthesis and during fitting.

Physical synthesis optimizations that occur during the synthesis stage of the Quartus II compilation operate either on the output from another EDA synthesis tool or as an intermediate step in Quartus II integrated synthesis. These optimizations make changes to the synthesis netlist to improve either area or speed, depending on your selected optimization technique and effort level.

To view and modify the synthesis netlist optimization options, on the Assignments menu, click Settings. In the Category list, expand Compilation Process Settings and select Physical Synthesis Optimizations.

If you use a third-party EDA synthesis tool and want to determine if the Quartus II software can remap the circuit to improve performance, you can use the Perform WYSIWYG Primitive Resynthesis option. This option directs the Quartus II software to unmap the LEs in an atom netlist to logic gates and then map the gates back to Altera-specific primitives. Using Altera-specific primitives enables the Fitter to remap the circuits using architecture-specific techniques.

For more information, refer to Perform WYSIWYG Primitive Resynthesis logic option in Quartus II Help.

The Quartus II technology mapper optimizes the design to achieve maximum speed performance, minimum area usage, or balances high performance and minimal logic usage, according to the setting of the Optimization Technique option. Set this option to Speed or Balanced.

For more information, refer to Optimization Technique logic option in Quartus II Help.

The physical synthesis optimizations occur during the Fitter stage of the Quartus II compilation. Physical synthesis optimizations make placement-specific changes to the netlist that improve speed performance results for a specific Altera device.
The following physical synthesis optimizations are available during the Fitter stage for improving performance:

- Physical synthesis for combinational logic
- Automatic asynchronous signal pipelining
- Physical synthesis for registers
  - Register duplication
  - Register retiming

If you want the performance gain from physical synthesis only on parts of your design, you can apply the physical synthesis options on specific instances.

For more information, refer to Physical Synthesis Optimizations Page (Settings Dialog Box) in Quartus II Help.

To apply physical synthesis assignments for fitting on a per-instance basis, use the Quartus II Assignment Editor. The following assignments are available as instance assignments:

- Perform physical synthesis for combinational logic
- Perform register duplication for performance
- Perform register retiming for performance
- Perform automatic asynchronous signal pipelining

For information about making assignments, refer to Working With Assignments in the Assignment Editor in Quartus II Help.

**Turn Off Extra-Effort Power Optimization Settings**

If PowerPlay power optimization settings are set to Extra Effort, your design performance can be affected. If improving timing performance is more important than reducing power use, set the PowerPlay power optimization setting to Normal.

For more information, refer to PowerPlay Power Optimization logic option in Quartus II Help.

For more information about reducing power use, refer to the Power Optimization chapter of the Quartus II Handbook.

**Optimize Synthesis for Speed, Not Area**

The manner in which the design is synthesized has a large impact on design performance. Design performance varies depending on the way the design is coded, the synthesis tool used, and the options specified when synthesizing. Change your synthesis options if a large number of paths are failing or if specific paths are failing badly and have many levels of logic.

Set your device and timing constraints in your synthesis tool. Synthesis tools are timing-driven and optimized to meet specified timing requirements. If you do not specify a target frequency, some synthesis tools optimize for area.
Some synthesis tools offer an easy way to instruct the tool to focus on speed instead of area.

For more information, refer to *Optimization Technique logic option* in Quartus II Help

You can also specify this logic option for specific modules in your design with the Assignment Editor while leaving the default *Optimization Technique* setting at *Balanced* (for the best trade-off between area and speed for certain device families) or *Area* (if area is an important concern). You can also use the *Speed Optimization Technique for Clock Domains* option in the Assignment Editor to specify that all combinational logic in or between the specified clock domain(s) is optimized for speed.

To achieve best performance with push-button compilation, follow the recommendations in the following sections for other synthesis settings. You can use the DSE to experiment with different Quartus II synthesis options to optimize your design for the best performance.

For more information about the Design Space Explorer, refer to *About Design Space Explorer* in Quartus II Help.

**Flatten the Hierarchy During Synthesis**

Synthesis tools typically let you preserve hierarchical boundaries, which can be useful for verification or other purposes. However, the best optimization results generally occur when the synthesis tool optimizes across hierarchical boundaries, because doing so often allows the synthesis tool to perform the most logic minimization, which can improve performance. Whenever possible, flatten your design hierarchy to achieve the best results. If you are using Quartus II incremental compilation, you cannot flatten your design across design partitions. Incremental compilation always preserves the hierarchical boundaries between design partitions. Follow Altera’s recommendations for design partitioning, such as registering partition boundaries to reduce the effect of cross-boundary optimizations.

For more information about using incremental compilation and recommendations for design partitioning, refer to the *Quartus II Incremental Compilation for Hierarchical and Team-Based Design* chapter of the *Quartus II Handbook*.

**Set the Synthesis Effort to High**

Some synthesis tools offer varying synthesis effort levels to trade off compilation time with synthesis results. Set the synthesis effort to *high* to achieve best results when applicable.
Change State Machine Encoding
State machines can be encoded using various techniques. One-hot encoding, which uses one register for every state bit, usually provides the best performance. If your design contains state machines, changing the state machine encoding to one-hot can improve performance at the cost of area.

For more information, refer to State Machine Processing logic option in Quartus II Help.

Duplicate Logic for Fan-Out Control
Duplicating logic or registers can help improve timing in cases where moving a register in a failing timing path to reduce routing delay creates other failing paths or where there are timing problems due to the fan-out of the registers. Most often, timing failures occur not because of the high fan-out registers, but because of the location of those registers. Duplicating registers, where source and destination registers are physically close, can help improve slack on critical paths.

Many synthesis tools support options or attributes that specify the maximum fan-out of a register. When using Quartus II integrated synthesis, you can set the Maximum Fan-Out logic option in the Assignment Editor to control the number of destinations for a node so that the fan-out count does not exceed a specified value. You can also use the maxfan attribute in your HDL code. The software duplicates the node as required to achieve the specified maximum fan-out.

Logic duplication using Maximum Fan-Out assignments normally increases resource utilization and can potentially increase compilation time, depending on the placement and the total resource usage within the selected device. The improvement in timing performance that results because of Maximum Fan-Out assignments is very design-specific. This is because when you use the Maximum Fan-Out assignment, although the Fitter duplicates the source logic to limit the fan-out, it may not be able to control the destinations that each of the duplicated sources drive. Since the Maximum Fan-Out destination does not specify which of the destinations the duplicated source should drive, it is possible that it might still be driving logic located all around the device. To avoid this situation, you could use the Manual Logic Duplication logic option.

If you are using Maximum Fan-Out assignments, Altera recommends benchmarking your design with and without these assignments to evaluate whether they give the expected improvement in timing performance. Use the assignments only when you get improved results.

You can manually duplicate registers in the Quartus II software regardless of the synthesis tool used. To duplicate a register, apply the Manual Logic Duplication logic option to the register with the Assignment Editor.

Various Fitter optimizations may cause a small violation to the Maximum Fan-Out assignments to improve timing.

For more information, refer to Manual Logic Duplication logic option in Quartus II Help.
Prevent Shift Register Inference

In some cases, turning off the inference of shift registers increases performance. Doing so forces the software to use logic cells to implement the shift register instead of implementing the registers in memory blocks using the ALTSHIFT_TAPS megafunction. If you implement shift registers in logic cells instead of memory, logic utilization is increased.

Use Other Synthesis Options Available in Your Synthesis Tool

With your synthesis tool, experiment with the following options if they are available:

- Turn on register balancing or retiming
- Turn on register pipelining
- Turn off resource sharing

These options can increase performance, but typically increase the resource utilization of your design.

Fitter Seed

The Fitter seed affects the initial placement configuration of the design. Changing the seed value changes the Fitter results because the fitting results change whenever there is a change in the initial conditions. Each seed value results in a somewhat different fit, and you can experiment with several different seeds to attempt to obtain better fitting results and timing performance.

When there are changes in your design, there is some random variation in performance between compilations. This variation is inherent in placement and routing algorithms—there are too many possibilities to try them all and get the absolute best result, so the initial conditions change the compilation result.

Any design change that directly or indirectly affects the Fitter has the same type of random effect as changing the seed value. This includes any change in source files, Analysis & Synthesis Settings, Fitter Settings, or Timing Analyzer Settings. The same effect can appear if you use a different computer processor type or different operating system, because different systems can change the way floating point numbers are calculated in the Fitter.

If a change in optimization settings slightly affects the register-to-register timing or number of failing paths, you cannot always be certain that your change caused the improvement or degradation, or whether it could be due to random effects in the Fitter. If your design is still changing, running a seed sweep (compiling your design with multiple seeds) determines whether the average result has improved after an optimization change and whether a setting that increases compilation time has benefits worth the increased time (such as setting the Physical Synthesis Effort to Extra). The sweep also shows the amount of random variation to expect for your design.

If your design is finalized, you can compile your design with different seeds to obtain one optimal result. However, if you subsequently make any changes to your design, you might need to perform seed sweep again.

On the Assignments menu, select Fitter Settings to control the initial placement with the seed. You can use the DSE to perform a seed sweep easily.
You can use the following Tcl command from a script to specify a Fitter seed:

```tcl
set_global_assignment -name SEED <value>
```

For more information about compiling your design with different seeds using the Design Space Explorer (DSE seed sweep), refer to About Design Space Explorer in Quartus II Help.

**Set Maximum Router Timing Optimization Level**

To improve routability in designs where the router did not pick up the optimal routing lines, set the **Router Timing Optimization Level** to **Maximum**. This setting determines how aggressively the router tries to meet the timing requirements. Setting this option to **Maximum** can increase design speed slightly at the cost of increased compilation time. Setting this option to **Minimum** can reduce compilation time at the cost of slightly reduced design speed. The default value is **Normal**.

For more information, refer to Router Timing Optimization Level logic option in Quartus II Help.

**LogicLock Assignments**

Using LogicLock assignments to improve timing performance is only recommended for older Altera devices, such as the MAX II family. For other device families, especially for larger devices such as Arria and Stratix series devices, Altera does not recommend using LogicLock assignments to improve timing performance. For these devices, use the LogicLock feature for performance preservation and to floorplan your design.

LogicLock assignments do not always improve the performance of the design. In many cases, you cannot improve upon results from the Fitter by making location assignments. If there are existing LogicLock assignments in your design, remove the assignments if your design methodology permits it. Recompile the design, and then check if the assignments are making the performance worse.

When making LogicLock assignments, it is important to consider how much flexibility to give the Fitter. LogicLock assignments provide more flexibility than hard location assignments. Assignments that are more flexible require higher Fitter effort, but reduce the chance of design overconstraint. The following types of LogicLock assignments are available, listed in the order of decreasing flexibility:

- Auto size, floating location regions
- Fixed size, floating location regions
- Fixed size, locked location regions

For more information about using LogicLock regions, refer to the Analyzing and Optimizing the Design Floorplan with the Chip Planner chapter of the Quartus II Handbook.
If you are unsure of how big or where a LogicLock region should go, the **Auto/Floating** options are useful for your first pass. After you determine where a LogicLock region must go, modify the Fixed/Locked regions, as Auto/Floating LogicLock regions can hurt your overall performance. To determine what to put into a LogicLock region, refer to the timing analysis results and analyze the critical paths in the Chip Planner. The register-to-register timing paths in the Timing Analyzer section of the Compilation Report help you recognize patterns.

The following sections describe cases in which LogicLock regions can help to optimize a design.

**Hierarchy Assignments**

For a design with the hierarchy shown in Figure 12–6, which has failing paths in the timing analysis results similar to those shown in Table 12–3, mod_A is probably a problem module. In this case, a good strategy to fix the failing paths is to place the mod_A hierarchy block in a LogicLock region so that all the nodes are closer together in the floorplan.

![Figure 12–6. Design Hierarchy](image)

Table 12–3. Failing Paths in a Module Listed in Timing Analysis

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
</tr>
</thead>
<tbody>
<tr>
<td>mod_A</td>
<td>reg1</td>
</tr>
<tr>
<td>mod_A</td>
<td>reg3</td>
</tr>
<tr>
<td>mod_A</td>
<td>reg4</td>
</tr>
<tr>
<td>mod_A</td>
<td>reg7</td>
</tr>
<tr>
<td>mod_A</td>
<td>reg0</td>
</tr>
</tbody>
</table>

Hierarchical LogicLock regions are also important if you are using an incremental compilation flow. Place each design partition for incremental compilation in a separate LogicLock region to reduce conflicts and ensure good results as the design develops. You can use the auto size and floating location regions to find a good design floorplan, but fix the size and placement to achieve the best results in future compilations.
For more information about using incremental compilation and recommendations for creating a design floorplan using LogicLock regions, refer to the Quartus II Incremental Compilation for Hierarchical and Team-Based Design and Best Practices for Incremental Compilation and Floorplan Assignments chapters of the Quartus II Handbook, and Analyzing and Optimizing the Design Floorplan with the Chip Planner chapter of the Quartus II Handbook.

**Location Assignments**

If a small number of paths are failing to meet their timing requirements, you can use hard location assignments to optimize placement. Location assignments are less flexible for the Quartus II Fitter than LogicLock assignments. In some cases, when you are familiar with your design, you can enter location constraints in a way that produces better results.

Improving fitting results, especially for larger devices, such as Arria and Stratix series devices, can be difficult. Location assignments do not always improve the performance of the design. In many cases, you cannot improve upon the results from the Fitter by making location assignments.

**Metastability Analysis and Optimization Techniques**

Metastability problems can occur when a signal is transferred between circuitry in unrelated or asynchronous clock domains, because the designer cannot guarantee that the signal will meet its setup and hold time requirements. The mean time between failures (MTBF) is an estimate of the average time between instances when metastability could cause a design failure.

For more information about metastability and MTBF, refer to the Understanding Metastability in FPGAs white paper.

You can use the Quartus II software to analyze the average MTBF due to metastability when a design synchronizes asynchronous signals and to optimize the design to improve the MTBF. These metastability features are supported only for designs constrained with the TimeQuest analyzer, and for select device families.

If the MTBF of your design is low, refer to the Metastability Optimization section in the Timing Optimization Advisor, which suggests various settings that can help optimize your design in terms of metastability.

For details about the metastability features in the Quartus II software, refer to the Managing Metastability with the Quartus II Software chapter of the Quartus II Handbook. This chapter describes how to enable metastability analysis and identify the register synchronization chains in your design, provides details about metastability reports, and provides additional guidelines for managing metastability.
Design Evaluation for Timing Closure

Follow the guidelines in this section when you encounter timing failures in a design. The guidelines show you how to evaluate compilation results of a design and how to address some of the problems. While the guideline does not cover specific examples of restructuring RTL to improve design speed, the analysis techniques help you to evaluate changes that may have to be made to RTL to close timing.

Review Compilation Results

Review Messages

After compiling your design, review the messages in each section of the compilation report. Most designs that fail timing start out with other problems that are reported as warning messages during compilation. Determine what causes a warning message, and whether the warning should be fixed or ignored. After reviewing the warning messages, review the informational messages. Take note of anything unexpected, for example, unconnected ports, ignored constraints, missing files, and assumptions or optimizations that the software made.

Evaluate Physical Synthesis Results

If physical synthesis is enabled, the software can duplicate and retime registers, and modify combinatorial logic during synthesis. After compilation, review the Optimization Results reports in the Analysis & Synthesis section. The reports list the optimizations performed by the physical synthesis optimizations, such as register duplication, retiming, and removal. These reports can be found in the Compilation Report panel (Figure 12–7).

Figure 12–7. Optimization Results reports
When physical synthesis is enabled, compilation messages include a summary of the physical synthesis algorithms that were run, the performance improvement each algorithm achieved, and the elapsed time. The reported improvement is the sum of the largest improvement estimated to be achievable in each timing-critical clock domain. The values for the slack improvements can vary between compiles because of the random starting point of the compilation algorithms, but the values should be similar. Figure 12–8 shows an example of the messages.

**Figure 12–8. Compilation Messages**

- Starting physical synthesis algorithm register retiming
- Physical synthesis algorithm register retiming complete, estimated slack improvement of 657 ps
- Starting physical synthesis algorithm combinational resynthesis using boolean division
- Physical synthesis algorithm combinational resynthesis using boolean division complete, estimated slack improvement of 471 ps
- Starting physical synthesis algorithm register retiming
- Physical synthesis algorithm register retiming complete, estimated slack improvement of 0 ps
- Starting physical synthesis algorithm combinational resynthesis using boolean division
- Physical synthesis algorithm combinational resynthesis using boolean division complete, estimated slack improvement of 0 ps
- Physical synthesis optimizations for speed complete; elapsed CPU time is 00:33:42

---

**Evaluate Fitter Netlist Optimizations**

The Fitter can also perform netlist optimizations to the design netlist. Major changes include register packing, duplicating or deleting logic cells, retiming registers, inverting signals, or modifying nodes in a general way such as moving an input from one logic cell to another. These reports can be found in the Netlist Optimizations results of the Fitter section, and they should also be reviewed.

---

**Evaluate optimization results**

After checking what optimizations were done and how they improved performance, evaluate the runtime it took to get the extra performance. To reduce compilation time, review the physical synthesis and netlist optimizations over a couple of compilations, and edit the RTL to reflect the changes that physical synthesis performed. If a particular set of registers consistently get retimed, edit the RTL to retime the registers the same way. If the changes are made to match what the physical synthesis algorithms did, the physical synthesis options can be turned off to save compile time while getting the same type of performance improvement.

---

**Evaluate resource usage**

Evaluate a variety of resources used in the design, including global and non-global signal usage, routing utilization, and clustering difficulty.
Global and non-global usage

If your design contains a lot of clocks, evaluate global and non-global signals. Determine whether global resources are being used effectively, and if not, consider making changes. These reports can be found in the Resource Section under Fitter in the Compilation Report panel. Figure 12–9 shows an example of inefficient use of a global clock. The highlighted line has a single fan-out from a global clock. Assigning it to a Regional Clock would make the Global Clock available for another signal. You can ignore signals with an empty value in the Global Line Name column as the signal uses dedicated routing, and not a clock buffer.

Figure 12–9. Inefficient Use of a Global Clock

<table>
<thead>
<tr>
<th>Global &amp; Other Fast Signals</th>
<th>Fan-Out</th>
<th>Global Resource Used</th>
<th>Global Line Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>FRACTIONAL_PLL_X98_Y2_N0</td>
<td>1</td>
<td>Global Clock</td>
<td></td>
</tr>
<tr>
<td>PLLOUTPUTCOUNTER_X98_Y2_N1</td>
<td>29044</td>
<td>Global Clock</td>
<td>GCLK7</td>
</tr>
<tr>
<td>PLLOUTPUTCOUNTER_X98_Y13_N1</td>
<td>253103</td>
<td>Global Clock</td>
<td>GCLK6</td>
</tr>
<tr>
<td>FF_X165_756_N1</td>
<td>280349</td>
<td>Global Clock</td>
<td>GCLK8</td>
</tr>
<tr>
<td>PIN_AE17</td>
<td>4607</td>
<td>Global Clock</td>
<td>GCLK4</td>
</tr>
<tr>
<td>FRACTIONAL_PLL_X98_Y11_N0</td>
<td>1</td>
<td>Global Clock</td>
<td></td>
</tr>
<tr>
<td>PLLOUTPUTCOUNTER_X98_Y3_N1</td>
<td>1</td>
<td>Global Clock</td>
<td>GCLK5</td>
</tr>
<tr>
<td>PLLOUTPUTCOUNTER_X98_Y13_N1</td>
<td>1691</td>
<td>Regional Clock</td>
<td>RCLK29</td>
</tr>
<tr>
<td>PLLOUTPUTCOUNTER_X98_Y8_N1</td>
<td>302</td>
<td>Regional Clock</td>
<td>RCLK23</td>
</tr>
<tr>
<td>PLLOUTPUTCOUNTER_X98_Y11_N1</td>
<td>141</td>
<td>Regional Clock</td>
<td>RCLK25</td>
</tr>
<tr>
<td>PLLOUTPUTCOUNTER_X98_Y10_N1</td>
<td>17</td>
<td>Regional Clock</td>
<td>RCLK22</td>
</tr>
</tbody>
</table>

The Non-Global High Fan-Out Signals report lists the highest fan-out nodes that are not routed on global signals. Reset and enable signals are at the top of the list. If there is routing congestion in the design, and there are high fan-out non-global nodes in the congested area, consider using global or regional signals to fan-out the nodes, or duplicate the high fan-out registers so that each of the duplicates can have fewer fan-outs. Use the Chip Planner to locate high fan-out nodes, to report routing congestion, and to determine whether the alternatives are viable.

Routing usage

Review routing usage reported in the Fitter Resource Usage Summary report. Figure 12–10 shows an example of the report.

Figure 12–10. Fitter Resource Usage Summary Report

<table>
<thead>
<tr>
<th>Resource Usage Summary</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>43</td>
<td>12 / 36</td>
</tr>
<tr>
<td>44</td>
<td>12 / 36</td>
</tr>
<tr>
<td>45</td>
<td>12 / 36</td>
</tr>
<tr>
<td>46</td>
<td>1.64 / 75 %</td>
</tr>
<tr>
<td>47</td>
<td>55% / 55% / 55%</td>
</tr>
<tr>
<td>48</td>
<td>68% / 88% / 90%</td>
</tr>
</tbody>
</table>
average over 65% typically have difficulty meeting timing unless the RTL is well designed to tolerate a highly utilized chip. Peak values at or above 90% are likely to have problems with timing closure; a 100% peak value indicates that all routing in an area of the device has been used, so there is a high possibility of degradation in timing performance. Figure 12–11 shows the Report Routing Utilization report.

**Figure 12–11. Report Routing Utilization Report**

---

**Wires Added for Hold**

As part of the fitting process, the router can add wire between register paths to increase delay to meet hold time requirements. During the routing process, the router reports how much extra wire was used to meet hold time requirements. Excessive amounts of added wire can indicate problems with the constraint. Typically it would be caused by incorrect multicycle transfers, particularly between different rate clocks, and between different clock networks. The Fitter reports how much routing delay was
added in the Estimated Delay Added for Hold Timing report (Figure 12–12). Specific register paths can be reviewed to view whether a delay was added to meet hold requirements.

**Figure 12–12. Estimated Delay Added for Hold Timing Report**

![Table of Contents](image)

An example of an incorrect constraint which can cause the router to add wire for hold requirements is when there is data transfer from 1x to 2x clocks. Assume the design intent is to allow two cycles per transfer. Data can arrive any time in the two destination clock cycles by adding a multicycle setup constraint as shown in Example 12–4:

**Example 12–4. Multicycle Setup Constraint**

```
set_multicycle_path -from 1x -to 2x -setup -end 2
```
The timing requirement is relaxed by one 2x clock cycle, as shown in the black line in the waveform in Figure 12–13.

**Figure 12–13.**

![Waveform showing timing requirements](image)

However, the default hold requirement, shown with the dashed blue line, may cause the router to add wire to guarantee that data is delayed by one cycle. To correct the hold requirement, add a multicycle constraint with a hold option (Example 12–5).

**Example 12–5. Multicycle Constraint with a Hold Option**

```plaintext
set_multicycle_path -from 1x -to 2x -setup -end 2
set_multicycle_path -from 1x -to 2x -hold -end 1
```

The orange dashed line in Figure 12–13 represents the hold relationship, and no extra wire is required to delay the data.

The router can also add wire for hold timing requirements when data is transferred in the same clock domain, but between clock branches that use different buffering. Transferring between clock network types happens more often between the periphery and the core. Figure 12–14 shows a case where data is coming into a device, and uses a periphery clock to drive the source register, and a global clock to drive the destination register. A global clock buffer has larger insertion delay than a periphery clock buffer. The clock delay to the destination register is much larger than to the source register, hence extra delay is necessary on the data path to ensure that it meets its hold requirement.

**Figure 12–14.**

![Diagram showing clock network types](image)
To identify cases where a path has different clock network types, review the path in the TimeQuest timing analyzer, and check nodes along the source and destination clock paths. Also, check the source and destination clock frequencies to see whether they are the same, or multiples, and whether there are multicycle exceptions on the paths. In some cases, cross-domain paths may also be false by intent, so make sure there are false path exceptions on those.

If you suspect that routing is added to fix real hold problems, then disable the **Optimize hold timing** option (Figure 12–15). Recompile the design and rerun timing analysis to uncover paths that fail hold time.

**Figure 12–15. Optimize Hold Timing Option**

Disabling the **Optimize hold timing** option is a debug step, and should be left enabled (default state) during normal compiles. Wire added for hold is a normal part of timing optimization during routing and is not always a problem.

**Evaluate Other Reports and Adjust Settings Accordingly**

**Difficulty Packing Design**

In the Fitter Resource Section, under the **Resource Usage Summary**, review the **Difficulty Packing Design** report. The **Difficulty Packing Design** report reports the effort level (low, medium, or high) of the Fitter to fit the design into the device, partition and LogicLock region. As the effort level of **Difficulty Packing Design** increases, timing closure gets harder. Going from medium to high can result in significant drop in performance or increase in compile time. Consider reducing logic to reduce packing difficulty.

**Review Ignored Assignments**

The **Compilation Report** includes details of any assignments ignored by the Fitter. Assignments typically get ignored if design names change, but assignments are not updated. Make sure any intended assignments are not being ignored.

**Review Non-Default Settings**

The reports from Synthesis and Fitter show non-default settings used in a compilation. Review the non-default settings to ensure the design benefits from the change.
Review Floorplan

Use the Chip Planner for reviewing placement. The Chip Planner can be used to locate hierarchical entities, and colors each located entity in the floorplan. Look for logic that seems out of place, based on where you would expect it to be. For example, logic that interfaces with I/Os should be close to the I/Os, and logic that interfaces with an IP or memory must be close to the IP or memory. Figure 12–16 shows an example of a floorplan with color-coded entities. In the floorplan, the green block is spread apart. Check to see if those paths are failing timing, and if so, what connects to that module that could affect placement. The blue and aqua blocks are spread out and mixed together. Check and see if there are many connections between the two modules that may contribute to this. The pink logic at the bottom should interface with I/Os at the bottom edge.

Figure 12–16. Floorplan with Color-Coded Entities

Check fan-in and fan-out of a highlighted module by using the buttons on the task bar shown in Figure 12–17:

Figure 12–17. Fan-in and Fan-Out Buttons

Look for signals that go a long way across the chip and see if they are contributing to timing failures.
Check global signal usage for signals that may affect logic placement. Logic feeding a global buffer may be pulled close to the buffer, away from related logic. High fan-out on non-global resource may pull logic together.

Check for routing congestion. Highly congested areas may cause logic to be spread out, and the design may be difficult to route.

**Evaluate Placement and Routing**

Review duration of parts of compile time in Fitter messages. If routing takes much more time than placement, then meeting timing may be more difficult than the placer predicted.

**Adjust Placement Effort**

Increasing the Placement Effort Multiplier to improve placement quality may be a good tradeoff at the cost of higher compile time, but the benefit is design dependent. The value should be adjusted after reviewing and optimizing other settings and RTL. Try an increased value, up to 4, and reset to default if performance or compile time does not improve.

![Figure 12–18. Placement Effort Multiplier](image-url)
Adjust Fitter Effort

To increase effort, enable the Standard Fit (highest effort) option. The default Auto Fit option reduces Fitter effort when it estimates timing requirements are met.

Figure 12–19.

Review Timing Constraints

Ensure that clocks are constrained with the correct frequency requirements. Using derive_pll_clocks assignment keeps generated clock settings updated. TimeQuest can be useful in reviewing SDC constraints. For example, under Diagnostic in the Task panel, the Report Ignored Constraints report reports any incorrect names in the design, most commonly caused by changes in the design hierarchy. Use the Report Unconstrained Paths report to locate unconstrained paths. Add constraints as necessary so that the design can be optimized.
Review Details of Timing Paths

Show Timing Path Routing

Showing routing for a path can help uncover unusual routing delays. In the TimeQuest Tasks panel, enable the Report panel name option, and then select Report Timing. Then, turn on the Show routing option to show routing wires in the path (Figure 12–20).

Figure 12–20.

The Extra Fitter Information tab shows a miniature floorplan with the path highlighted. The path can also be located in the Chip Planner for viewing routing congestion, and to view whether nodes in a path are placed close together or far apart.

Global Network Buffers

A routing path can be used to identify global network buffers that fail timing. Buffer locations are named according to the network they drive:

- CLK_CTRL_Gn—for Global driver
- CLK_CTRL_Rn—for Regional driver

Buffers to access the global networks are located in the center of each side of the device. The buffering to route a core logic signal on a global signal network will cause insertion delay. Some trade offs to consider for global and non-global routing are source location, insertion delay, fan-out, distance a signal travels, and possible congestion if the signal is demoted to local routing.

Source Location

If the register feeding the global buffer cannot be moved closer, then consider changing either the design logic or the routing type.
**Insertion delay**

If a global signal is required, consider adding half a cycle to timing by using a negative-edge triggered register to generate the signal (Figure 12–21) and use a multicycle setup constraint (Figure 12–22).

**Fan-Out**

Nodes with very high fan-out that use local routing tend to pull logic that they drive close to the source node. This can make other paths fail timing. Duplicating registers can help reduce the impact of high fan-out paths. Consider manually duplicating and preserving these registers. Using a `MAX_FANOUT` assignment may make arbitrary groups of fan-out nodes, whereas a designer can make more intelligent fan-out groups.

**Global Networks**

If a signal should use a different type of global signal than it has automatically been assigned, use the Global Signal assignment to control the global signal usage on a per-signal basis. For example, if local routing is desired, set the Global Signal assignment to `OFF` (Figure 12–23).

<table>
<thead>
<tr>
<th>To</th>
<th>Assignment Name</th>
<th>Value</th>
<th>Enabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>reg clk</td>
<td>Global Signal</td>
<td>Off</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Resets and Global Networks

Reset signals are often routed on global networks. Sometimes, the use of a global network causes recovery failures. Consider reviewing the placement of the register that generates the reset and the routing path of the signal.

Suspicious Setup

Suspicious setup failures include paths with very small or very large requirements. One typical cause is math precision error. For example, 10Mhz/3 = 33.33 ns per period. In three cycles, the time would be 99.999 ns vs 100.000 ns. Setting a maximum delay could provide an appropriate setup relationship.

Another cause of failure would be paths that should be false by design intent, such as:

- asynchronous paths that are handled through FIFOs, or
- slow asynchronous paths that rely on handshaking for data that remain available for multiple clock cycles.

To prevent the Fitter from having to meet unnecessarily restrictive timing requirements, consider adding false or multicycle path statements.

Logic Depth

The Statistics tab in the TimeQuest path report shows the levels of logic in a path. If the path fails timing and the number of logic levels is high, consider adding pipelining in that part of the design.

Auto Shift Register Replacement

Shift registers or register chains can be converted to RAM during synthesis to save area. However, conversion to RAM often reduces speed. The names of the converted registers will include "altshift_taps".

If paths that fail timing begin or end in shift registers, consider disabling the Auto Shift Register Replacement option. Registers that are intended for pipelining should not be converted. For shift registers that are converted to a chain, evaluate area/speed trade off of implementing in RAM or logic cells. If a design is close to full, shift register conversion to RAM may benefit non-critical clock domains by saving area. The settings can be changed globally or on a register or hierarchy basis from the default of AUTO to OFF.
Clocking Architecture

Review the clock region boundaries in the Chip Planner. You must place registers driven by a regional clock in one quadrant of the chip.

Figure 12–24.

Timing failure can occur when the I/O interface at the top of the device connects to logic driven by a regional clock which is in one quadrant of the device, and placement restrictions force long paths to and from some of the I/Os to logic across quadrants.

Use a different type of clock source to drive the logic - global, which covers the whole device, or dual-regional which covers half the device. Alternatively, you can reduce the frequency of the I/O interface to accommodate the long path delays. You can also redesign the pinout of the device to place all the specified I/Os adjacent to the regional clock quadrant. This issue can happen when register locations are restricted, such as with LogicLock regions, clocking resources, or hard blocks (memories, DSPs, IPs). The Extra Fitter Information tab in the TimeQuest report informs you when placement is restricted for nodes in a path.

Timing Closure Recommendations

The Report Timing Closure Recommendations task in the TimeQuest analyzer analyzes paths and provides specific recommendations based on path characteristics.
Making Adjustments and Recompiling

Look for obvious problems that you can fix with minimal effort. To identify where the Compiler had trouble meeting timing, perform seed sweeping with about five compiles. Doing so shows consistently failing paths. Consider recoding or redesigning that part of the design.

To reach timing closure, a well written RTL can be more effective than changing your compilation settings. Seed sweeping can also be useful if the timing failure is very small, and the design has already been optimized for performance improvements and is close to final release. Additionally, seed sweeping can be used for evaluating changes to compilation settings. Compilation results vary due to the random nature of fitter algorithms. If a compilation setting change produces lower average performance, undo the change.

Sometimes, settings or constraints can cause more problems than they fix. When significant changes to the RTL or design architecture have been made, compile periodically with default settings and without LogicLock regions, and re-evaluate paths that fail timing.

Partitioning often does not help timing closure, and should be done at the beginning of the design process. Adding partitions can increase logic utilization if it prevents cross-boundary optimizations, making timing closure harder and increasing compile times.

Adding LogicLock regions can be an effective part of timing closure, but must be done at the beginning of a design. Adding new LogicLock regions at the end of the design cycle can restrict placement, hence lowering the performance.

Scripting Support

You can run procedures and make settings described in this chapter in a Tcl script. You can also run some procedures at a command prompt. For detailed information about scripting command options, refer to the Quartus II command-line and Tcl API Help browser. To run the Help browser, type the following command at the command prompt:

```
quartus_sh --qhelp
```

For more information about Tcl scripting, refer to the Tcl Scripting chapter of the Quartus II Handbook. For more information about all settings and constraints in the Quartus II software, refer to the Quartus II Settings File Manual. For more information about command-line scripting, refer to the Command-Line Scripting chapter of the Quartus II Handbook.

You can specify many of the options described in this section either in an instance, or at a global level, or both.

Use the following Tcl command to make a global assignment:

```
set_global_assignment -name <.gsf variable name> <value>
```

Use the following Tcl command to make an instance assignment:

```
set_instance_assignment -name <.gsf variable name> <value> -to <instance name>
```
If the <value> field includes spaces (for example, ‘Standard Fit’), you must enclose the value in straight double quotation marks.

**Initial Compilation Settings**

Use the Quartus II Settings File (.qsf) variable name in the Tcl assignment to make the setting along with the appropriate value. The Type column indicates whether the setting is supported as a global setting, an instance setting, or both.

Table 12–4 lists the .qsf variable name and applicable values for the settings described in the “Initial Compilation: Required Settings” section in the Design Optimization Overview chapter of the Quartus II Handbook. Table 12–5 lists the advanced compilation settings.

### Table 12–4. Initial Compilation Settings

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>.qsf File Variable Name</th>
<th>Values</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimize IOC Register Placement For Timing</td>
<td>OPTIMIZE_IOC_REGISTER_PLACEMENT_FOR_TIMING</td>
<td>ON, OFF</td>
<td>Global</td>
</tr>
<tr>
<td>Optimize Hold Timing</td>
<td>OPTIMIZE_HOLD_TIMING</td>
<td>OFF, IO PATHS AND MINIMUM TPD PATHS, ALL PATHS</td>
<td>Global</td>
</tr>
</tbody>
</table>

### Table 12–5. Advanced Compilation Settings

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>.qsf File Variable Name</th>
<th>Values</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Router Timing Optimization level</td>
<td>ROUTER_TIMING_OPTIMIZATION_LEVEL</td>
<td>NORMAL, MINIMUM, MAXIMUM</td>
<td>Global</td>
</tr>
</tbody>
</table>

**Resource Utilization Optimization Techniques (LUT-Based Devices)**

Table 12–6 lists the .qsf file variable name and applicable values for the settings described in “Optimizing Timing (LUT-Based Devices)” on page 12–10.

### Table 12–6. Resource Utilization Optimization Settings (Part 1 of 2)

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>.qsf File Variable Name</th>
<th>Values</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Auto Packed Registers</td>
<td>AUTO_PACKED_REGISTERS_&lt;device family name&gt;</td>
<td>OFF, NORMAL, MINIMIZE AREA, MINIMIZE AREA WITH CHAINS, AUTO</td>
<td>Global, Instance</td>
</tr>
<tr>
<td>Perform WYSIWYG Primitive Resynthesis</td>
<td>ADV_NETLIST_OPT_SYNTH_WYSIWYG_REMAP</td>
<td>ON, OFF</td>
<td>Global, Instance</td>
</tr>
<tr>
<td>Physical Synthesis for Combinational Logic for Reducing Area</td>
<td>PHYSICAL_SYNTHESIS_COMBO_LOGIC_FOR_AREA</td>
<td>ON, OFF</td>
<td>Global, Instance</td>
</tr>
<tr>
<td>Physical Synthesis for Mapping Logic to Memory</td>
<td>PHYSICAL_SYNTHESIS_MAP_LOGIC_TO_MEMORY_FOR_AREA</td>
<td>ON, OFF</td>
<td>Global, Instance</td>
</tr>
<tr>
<td>Optimization Technique</td>
<td>&lt;device family name&gt;_OPTIMIZATION_TECHNIQUE</td>
<td>AREA, SPEED, BALANCED</td>
<td>Global, Instance</td>
</tr>
</tbody>
</table>
Table 12–6. Resource Utilization Optimization Settings  (Part 2 of 2)

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>.qsf File Variable Name</th>
<th>Values</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed Optimization Technique for Clock Domains</td>
<td>SYNTH_CRITICAL_CLOCK</td>
<td>ON, OFF</td>
<td>Instance</td>
</tr>
<tr>
<td>State Machine Encoding</td>
<td>STATE_MACHINE_PROCESSING</td>
<td>AUTO, ONE-HOT, GRAY, JOHNSON, MINIMAL BITS, ONE-HOT, SEQUENTIAL, USER-ENCODER</td>
<td>Global, Instance</td>
</tr>
<tr>
<td>Auto RAM Replacement</td>
<td>AUTO_RAM_RECOGNITION</td>
<td>ON, OFF</td>
<td>Global, Instance</td>
</tr>
<tr>
<td>Auto ROM Replacement</td>
<td>AUTO_ROM_RECOGNITION</td>
<td>ON, OFF</td>
<td>Global, Instance</td>
</tr>
<tr>
<td>Auto Shift Register Replacement</td>
<td>AUTO_SHIFT_REGISTER_RECOGNITION</td>
<td>ON, OFF</td>
<td>Global, Instance</td>
</tr>
<tr>
<td>Auto Block Replacement</td>
<td>AUTO_DSP_RECOGNITION</td>
<td>ON, OFF</td>
<td>Global, Instance</td>
</tr>
<tr>
<td>Number of Processors for Parallel Compilation</td>
<td>NUM_PARALLEL_PROCESSORS</td>
<td>Integer between 1 and 16 inclusive, or ALL</td>
<td>Global</td>
</tr>
</tbody>
</table>

Note to Table 12–6:
(1) Allowed values for this setting depend on the device family that you select.

I/O Timing Optimization Techniques (LUT-Based Devices)

Table 12–7 lists the .qsf file variable name and applicable values for the I/O timing optimization settings.

Table 12–7. I/O Timing Optimization Settings

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>.qsf File Variable Name</th>
<th>Values</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimize IOC Register Placement For Timing</td>
<td>OPTIMIZE_IOC_REGISTER_PLACEMENT_FOR_TIMING</td>
<td>ON, OFF</td>
<td>Global</td>
</tr>
<tr>
<td>Fast Input Register</td>
<td>FAST_INPUT_REGISTER</td>
<td>ON, OFF</td>
<td>Instance</td>
</tr>
<tr>
<td>Fast Output Register</td>
<td>FAST_OUTPUT_REGISTER</td>
<td>ON, OFF</td>
<td>Instance</td>
</tr>
<tr>
<td>Fast Output Enable Register</td>
<td>FAST_OUTPUT_ENABLE_REGISTER</td>
<td>ON, OFF</td>
<td>Instance</td>
</tr>
<tr>
<td>Fast OCT Register</td>
<td>FAST_OCT_REGISTER</td>
<td>ON, OFF</td>
<td>Instance</td>
</tr>
</tbody>
</table>
### Register-to-Register Timing Optimization Techniques (LUT-Based Devices)

Table 12–8 lists the .qsf file variable name and applicable values for the settings described in “Register-to-Register Timing Optimization Techniques (LUT-Based Devices)” on page 12–17.

<table>
<thead>
<tr>
<th>Setting Name</th>
<th>.qsf File Variable Name</th>
<th>Values</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perform WYSIWYG Primitive Resynthesis</td>
<td>ADV_NETLIST_OPT_SYNTH_WYSIWYG_REMAP</td>
<td>ON, OFF</td>
<td>Global, Instance</td>
</tr>
<tr>
<td>Perform Physical Synthesis for Combinational Logic</td>
<td>PHYSICAL_SYNTHESIS_COMBO_LOGIC</td>
<td>ON, OFF</td>
<td>Global, Instance</td>
</tr>
<tr>
<td>Perform Register Duplication</td>
<td>PHYSICAL_SYNTHESIS_REGISTER_DUPLICATION</td>
<td>ON, OFF</td>
<td>Global, Instance</td>
</tr>
<tr>
<td>Perform Register Retiming</td>
<td>PHYSICAL_SYNTHESIS_REGISTER_RETIMING</td>
<td>ON, OFF</td>
<td>Global, Instance</td>
</tr>
<tr>
<td>Perform Automatic Asynchronous Signal Pipelining</td>
<td>PHYSICAL_SYNTHESISASYNCHRONOUS_SIGNAL_PIPELINING</td>
<td>ON, OFF</td>
<td>Global, Instance</td>
</tr>
<tr>
<td>Physical Synthesis Effort</td>
<td>PHYSICAL_SYNTHESIS_EFFORT</td>
<td>NORMAL, EXTRA, FAST</td>
<td>Global</td>
</tr>
<tr>
<td>Fitter Seed</td>
<td>SEED</td>
<td>&lt;integer&gt;</td>
<td>Global</td>
</tr>
<tr>
<td>Maximum Fan-Out</td>
<td>MAX_FANOUT</td>
<td>&lt;integer&gt;</td>
<td>Instance</td>
</tr>
<tr>
<td>Manual Logic Duplication</td>
<td>DUPLICATE_ATOM</td>
<td>&lt;node name&gt;</td>
<td>Instance</td>
</tr>
<tr>
<td>Optimize Power during Synthesis</td>
<td>OPTIMIZE_POWER_DURING_SYNTHESIS</td>
<td>NORMAL, OFF EXTRA_EFFORT</td>
<td>Global</td>
</tr>
<tr>
<td>Optimize Power during Fitting</td>
<td>OPTIMIZE_POWER_DURING_FITTING</td>
<td>NORMAL, OFF EXTRA_EFFORT</td>
<td>Global</td>
</tr>
</tbody>
</table>
Document Revision History

Table 12–9 lists the revision history for this chapter.

Table 12–9. Document Revision History (Part 1 of 2)

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>■ Removed Optimizing Timing (Macrocell-Based CPLDs) section.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Updated “Optimize Multi-Corner Timing” on page 12–2, and “Fitter Aggressive Routability Optimization” on page 12–3.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Updated “Timing Analysis with the TimeQuest Timing Analyzer” on page 12–4 to show how to access the Report All Summaries command.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Updated “Ignored Timing Constraints” on page 12–3 to include a help link to Fitter Summary Reports with the Ignored Assignment Report information.</td>
</tr>
<tr>
<td>May 2013</td>
<td>13.0.0</td>
<td>■ Renamed chapter title from Area and Timing Optimization to “Timing Closure and Optimization”</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Removed design and area/resources optimization information.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Added the following sections:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ “Fitter Aggressive Routability Optimization” on page 12–3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ “Tips for Analyzing Paths from/to the Source and Destination of Critical Path” on page 12–7,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ “Tips for Locating Multiple Paths to the Chip Planner” on page 12–8,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ “Tips for Creating a .tcl Script to Monitor Critical Paths Across Compiles” on page 12–9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Minor text edits throughout the chapter.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Updated Table 13–6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Added the “Spine Clock Limitations” section</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Removed the Change State Machine Encoding section from page 19</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Removed Figure 13-5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Minor text edits throughout the chapter.</td>
</tr>
</tbody>
</table>
### Table 12–9. Document Revision History (Part 2 of 2)

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Changes</th>
</tr>
</thead>
</table>
| May 2011   | 11.0.0  | ■ Reorganized sections in “Initial Compilation: Optional Fitter Settings” section  
■ Added new information to “Resource Utilization” section  
■ Added new information to “Duplicate Logic for Fan-Out Control” section  
■ Added links to Help  
■ Additional edits and updates throughout chapter |
| December 2010 | 10.1.0  | ■ Added links to Help  
■ Updated device support  
■ Added “Debugging Timing Failures in the TimeQuest Analyzer” section  
■ Removed Classic Timing Analyzer references  
■ Other updates throughout chapter |
| August 2010 | 10.0.1  | Corrected link |
| July 2010  | 10.0.0  | ■ Moved Compilation Time Optimization Techniques section to new *Reducing Compilation Time* chapter  
■ Removed references to Timing Closure Floorplan  
■ Moved Smart Compilation Setting and Early Timing Estimation sections to new *Reducing Compilation Time* chapter  
■ Added Other Optimization Resources section  
■ Removed outdated information  
■ Changed references to DSE chapter to Help links  
■ Linked to Help where appropriate  
■ Removed Referenced Documents section |

For previous versions of the *Quartus II Handbook*, refer to the *Quartus II Handbook Archive.*