Section III. Area, Timing, Power, and Compilation Time Optimization

This section introduces features in the Quartus® II software that you can use to optimize area, timing, power, and compilation time when you design for programmable logic devices (PLDs).

This section includes the following chapters:

■ Chapter 11, Design Optimization Overview
  This chapter summarizes features in the Quartus II software that you can use to achieve the highest design performance when you design for PLDs, especially high density FPGAs.

■ Chapter 12, Reducing Compilation Time
  This chapter describes techniques for reducing the amount of time it takes to compile and recompile your design, accelerating your design process.

■ Chapter 13, Area and Timing Optimization
  This chapter describes a broad spectrum of Quartus II software features and design techniques to reduce resource usage and improve timing performance when designing for Altera® devices. This chapter also explains how and when to use some of the features described in other chapters of the Quartus II Handbook.

■ Chapter 14, Power Optimization
  This chapter describes the power-driven compilation feature and flow in detail, as well as low power design techniques that can further reduce power consumption in your design.

■ Chapter 15, Analyzing and Optimizing the Design Floorplan with the Chip Planner
  You can use the Chip Planner to perform design analysis and create a design floorplan. This chapter discusses how to analyze and optimize the design floorplan with the Chip Planner.

■ Chapter 16, Netlist Optimizations and Physical Synthesis
  This chapter explains how the physical synthesis optimizations in the Quartus II software can improve your quality of results. This chapter also provides information about preserving and writing out a new netlist, and provides guidelines for applying the various options.
11. Design Optimization Overview

This chapter introduces features in Altera’s Quartus® II software that you can use to achieve the highest design performance when you design for programmable logic devices (PLDs), especially high density FPGAs.

Introduction

Physical implementation can be an intimidating and challenging phase of the design process. The Quartus II software provides a comprehensive environment for FPGA designs, delivering unmatched performance, efficiency, and ease-of-use.

In a typical design flow, you must synthesize your design with Quartus II integrated synthesis or a third-party tool, place and route your design with the Fitter, and use the TimeQuest timing analyzer to ensure your design meets the timing requirements. With the PowerPlay Power Analyzer, you ensure the design’s power consumption is within limits.

Physical Implementation

Most optimization issues involve preserving previous results, reducing area, reducing critical path delay, reducing power consumption, and reducing runtime. The Quartus II software includes advisors to address each of these issues and helps you optimize your design. Run these advisors during physical implementation for advice about your specific design.

You can reduce the time spent on design iterations by following the recommended design practices for designing with Altera® devices. Design planning is critical for successful design timing implementation and closure.

For more information, refer to the Design Planning with the Quartus II Software chapter in volume 1 of the Quartus II Handbook.

Trade-Offs and Limitations

Many optimization goals can conflict with one another, so you might need to make trade-offs between different goals. For example, one major trade-off during physical implementation is between resource usage and critical path timing, because certain techniques (such as logic duplication) can improve timing performance at the cost of increased area. Similarly, a change in power requirements can result in area and timing trade-offs, such as if you reduce the number of high-speed tiles available, or if you attempt to shorten high-power nets at the expense of critical path nets.

In addition, system cost and time-to-market considerations can affect the choice of device. For example, a device with a higher speed grade or more clock networks can facilitate timing closure at the expense of higher power consumption and system cost.
Finally, not all designs can be realized in a hardware circuit with limited resources and
given constraints. If you encounter resource limitations, timing constraints, or power
constraints that cannot be resolved by the Fitter, consider rewriting parts of the HDL
code.

For more information, refer to the Area and Timing Optimization chapter in volume 2 of
the Quartus II Handbook.

Preserving Results and Enabling Teamwork

For some Quartus II Fitter algorithms, small changes to the design can have a large
impact on the final result. For example, a critical path delay can change by 10% or
more because of seemingly insignificant changes. If you are close to meeting your
timing objectives, you can use the Fitter algorithm to your advantage by changing the
fitter seed, which changes the pseudo-random result of the Fitter.

Conversely, if you cannot meet timing on a portion of your design, you can partition
that portion and prevent it from recompiling if an unrelated part of the design is
changed. This feature, known as incremental compilation, can reduce the Fitter
runtimes by up to 70% if the design is partitioned, such that only small portions
require recompilation at any one time.

When you use incremental compilation, you can apply design optimization options to
individual design partitions and preserve performance in other partitions by leaving
them untouched. Many optimization techniques often result in longer compilation
times, but by applying them only on specific partitions, you can reduce this impact
and complete iterations more quickly.

In addition, by physically floorplanning your partitions with LogicLock regions, you
can enable team-based flows and allow multiple people to work on different portions
of the design.

For more information, refer to Quartus II Incremental Compilation for Hierarchical and
Team-Based Designs in volume 1 of the Quartus II Handbook and About Incremental
Compilation in Quartus II Help.

Reducing Area

By default, the Quartus II Fitter might physically spread a design over the entire device
to meet the set timing constraints. If you prefer to optimize your design to use the
smallest area, you can change this behavior. If you require reduced area, you can
enable certain physical synthesis options to modify your netlist to create a more
area-efficient implementation, but at the cost of increased runtime and decreased
performance.

For more information, refer to the Area and Timing Optimization and Netlist
Optimizations and Physical Synthesis chapters in volume 2 and the Recommended HDL
Coding Styles chapter in volume 1 of the Quartus II Handbook.
Reducing Critical Path Delay

To meet complex timing requirements involving multiple clocks, routing resources, and area constraints, the Quartus II software offers a close interaction between synthesis, timing analysis, floorplan editing, and place-and-route processes.

By default, the Quartus II Fitter tries to meet the specified timing requirements and stops trying when the requirements are met. Therefore, using realistic constraints is important to successfully close timing. If you under-constrain your design, you may get sub-optimal results. By contrast, if you over-constrain your design, the Fitter might over-optimize non-critical paths at the expense of true critical paths. In addition, you might incur an increased area penalty. Compilation time may also increase because of excessively tight constraints.

If your resource usage is very high, the Quartus II Fitter might have trouble finding a legal placement. In such circumstances, the Fitter automatically modifies some of its settings to try to trade off performance for area.

The Quartus II Fitter offers a number of advanced options that can help you improve the performance of your design when you properly set constraints. Use the Timing Optimization Advisor to determine which options are best suited for your design.

If you use incremental compilation, you can help resolve inter-partition timing requirements by locking down the results one partition at a time or by guiding the placement of the partitions with LogicLock regions. You might be able to improve the timing on such paths by placing the partitions optimally to reduce the length of critical paths. Once your inter-partition timing requirements are met, use incremental compilation to preserve the results and work on partitions that have not met timing requirements.

In high-density FPGAs, routing accounts for a major part of critical path timing. Because of this, duplicating or retiming logic can allow the Fitter to reduce delay on critical paths. The Quartus II software offers push-button netlist optimizations and physical synthesis options that can improve design performance at the expense of considerable increases of compilation time and area. Turn on only those options that help you keep reasonable compilation times and resource usage. Alternately, you can modify your HDL to manually duplicate or retime logic.

Reducing Power Consumption

The Quartus II software has features that help reduce design power consumption. The PowerPlay power optimization options control the power-driven compilation settings for Synthesis and the Fitter.

For more information, refer to the Power Optimization chapter in volume 2 of the Quartus II Handbook.

Reducing Runtime

Many Fitter settings influence compilation time. Most of the default settings in the Quartus II software are set for reduced compilation time. You can modify these settings based on your project requirements.
The Quartus II software supports parallel compilation in computers with multiple processors. This can reduce compilation times by up to 15% while giving the identical result as serial compilation.

You can also reduce compilation time with your iterations by using incremental compilation. Use incremental compilation when you want to change parts of your design, while keeping most of the remaining logic unchanged.

**Using Quartus II Tools**

The following sections describe several Quartus II tools that you can use to help optimize your design.

**Design Analysis**

The Quartus II software provides tools that help with a visual representation of your design. You can use the RTL Viewer to see a schematic representation of your design before synthesis and place-and-route. The Technology Map Viewer provides a schematic representation of the design implementation in the selected device architecture after synthesis and place-and-route. It can also include timing information.

With incremental compilation, the Design Partition Planner and the Chip Planner allow you to partition and layout your design at a higher level. In addition, you can perform many different tasks with the Chip Planner, including: making floorplan assignments, implementing engineering change orders (ECOs), and performing power analysis. Also, you can analyze your design and achieve a faster timing closure with the Chip Planner. The Chip Planner provides physical timing estimates, critical path display, and routing congestion view to help guide placement for optimal performance.

For more information, refer to the *Quartus II Incremental Compilation for Hierarchical and Team-Based Designs* and *Best Practices for Incremental Compilation Partitions and Floorplan Assignments* chapters in volume 1 and the *Engineering Change Management with the Chip Planner* chapter in volume 2 of the *Quartus II Handbook*.

**Advisors**

The Quartus II software includes several advisors to help you optimize your design and reduce compilation time. You can complete your design faster by following the recommendations in the Compilation Time Advisor, Incremental Compilation Advisor, Timing Optimization Advisor, Area Optimization Advisor, Resource Optimization Advisor, and Power Optimization Advisor. These advisors give recommendations based on your project settings and your design constraints.

For more information about advisors, refer to Quartus II Help.
Design Space Explorer

Use the Design Space Explorer (DSE) to find optimal settings in the Quartus II software. DSE automatically tries different combinations of netlist optimizations and advanced Quartus II software compiler settings, and reports the best settings for your design, based on your chosen primary optimization goal. You can try different seeds with the DSE if you are fairly close to meeting your timing or area requirements and find one seed that meets timing or area requirements. Finally, the DSE can run the different compilations on multiple computers in parallel, which shortens the timing closure process.

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

Conclusion

The Quartus II software includes a number of features and tools that you can use to optimize area, timing, power, and compilation time when you design for programmable logic devices (PLDs).

Document Revision History

Table 11–1 shows the revision history for this chapter.

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>December 2010</td>
<td>10.0.2</td>
<td>Changed to new document template. No change to content.</td>
</tr>
<tr>
<td>August 2010</td>
<td>10.0.1</td>
<td>Corrected link</td>
</tr>
<tr>
<td>July 2010</td>
<td>10.0.0</td>
<td>Initial release. Chapter based on topics and text in Section III of volume 2.</td>
</tr>
</tbody>
</table>

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

Take an online survey to provide feedback about this handbook chapter.
12. Reducing Compilation Time

The Quartus® II software offers several features and techniques to help reduce compilation time.

This chapter describes techniques to reduce compilation time when designing for Altera® devices, and includes the following topics:

- “Compilation Time Optimization Techniques”
- “Compilation Time Advisor” on page 12–2
- “Strategies to Reduce the Overall Compilation Time” on page 12–2
- “Reducing Synthesis Time and Synthesis Netlist Optimization Time” on page 12–5
- “Reducing Placement Time” on page 12–7
- “Reducing Routing Time” on page 12–8
- “Reducing Static Timing Analysis Time” on page 12–9
- “Setting Process Priority” on page 12–10

Compilation Time Optimization Techniques

The Analysis and Synthesis and Fitter modules require a lot of time. The Analysis and Synthesis module includes physical synthesis optimizations performed during synthesis, if you have turned on physical synthesis optimizations. The Fitter includes two steps, placement and routing, and also includes physical synthesis if you turned on the physical synthesis option with Normal or Extra effort levels. The Flow Elapsed Time section of the Compilation Report shows the duration of the Analysis and Synthesis and Fitter modules. The Fitter Messages report in the Fitter section of the Compilation Report shows the duration of placement and routing.

Placement is the process of finding optimum locations for the logic in your design. Placement includes Quartus II pre-Fitter operations, which place dedicated logic such as clocks, PLLs, and transceiver blocks. Routing is the process of connecting the nets between the logic in your design. Finding better placements for the logic in a design uses more compilation time. Good logic placement allows you to more easily meet your timing requirements and makes your design easier to route.

Example 12–1 shows the applicable messages with each time component in two-digit format, and days shown only if applicable:

Example 12–1.

<table>
<thead>
<tr>
<th>Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Info: Fitter placement operations ending: elapsed time =</td>
<td>days:hours:minutes:seconds</td>
</tr>
<tr>
<td>Info: Fitter routing operations ending: elapsed time =</td>
<td>days:hours:minutes:seconds</td>
</tr>
</tbody>
</table>
Example 12–2 shows an info message while the Fitter is running (including Placement and Routing). The Message window displays this message every hour to indicate Fitter operations are progressing normally.

Example 12–2.

Info: Placement optimizations have been running for 4 hour(s)

Compilation Time Advisor

A Compilation Time Advisor is available in the Quartus II software, which helps you to reduce compilation time. Run the Compilation Time Advisor on the Tools menu by pointing to Advisors and clicking Compilation Time Advisor. You can find all the compilation time optimizing techniques described in this section in the Compilation Time Advisor as well.

Strategies to Reduce the Overall Compilation Time

This section discusses strategies to reduce overall compilation time, including the following topics:

- “Using Parallel Compilation with Multiple Processors”
- “Using Incremental Compilation” on page 12–3
- “Using the Smart Compilation Setting” on page 12–4
- “Using Rapid Recompile” on page 12–4

Using Parallel Compilation with Multiple Processors

The Quartus II software can detect the number of processors available on a computer and use available processors to reduce compilation time. You can also control the number of processors used during a compilation on a per user basis. The Quartus II software can use up to 16 processors to run some algorithms in parallel and reduce compilation time. The Quartus II software turns on the parallel compilation by default to enable the software to detect available multiple processors. You can specify the maximum number of processors that the software can use if you want to reserve some of the available processors for other tasks.

Do not consider processors with Intel Hyper-Threading as more than one processor. If you have a single processor with Intel Hyper-Threading enabled, you should set the number of processors to one. Altera recommends that you do not use the Intel Hyper-Threading feature for Quartus II compilations, because it can increase runtimes.

The software does not necessarily use all the processors that you specify during a given compilation. Additionally, the software never uses more than the specified number of processors, enabling you to work on other tasks on your computer without it becoming slow or less responsive.

If you have partitioned your design and enabled parallel compilation, the Quartus II software can use different processors to compile those partitions simultaneously during the Analysis and Synthesis stage, resulting in high peak memory usage during Analysis and Synthesis.
By partitioning your design and allowing the Quartus II software to use two processors, you can reduce the compilation time by up to 10% on systems with two processing cores and by up to 20% on systems with four cores. With certain design flows in which timing analysis runs alone, using multiple processors can reduce the time required for timing analysis by an average of 10% when using two processors. This reduction can reach an average of 15% when using four processors.

You must partition your design to reduce compilation time successfully.

The actual reduction in compilation time depends on your design and on the specific compilation settings. For example, compilations with multi-corner optimization turned on benefit more from using multiple processors than do compilations that do not use multi-corner optimization. The runtime requirement is not reduced for some other compilation goals, such as Analysis and Synthesis. The Fitter (quartus_fit) and the Quartus II TimeQuest Timing Analyzer (quartus_sta) stages in the compilation can, in certain cases, benefit from the use of multiple processors. The Flow Elapsed Time panel of the Compilation Report shows the average number of processors for these stages. The Parallel Compilation panel of the appropriate report shows a more detailed breakdown of processor usage, such as the Fitter report. This panel is displayed only if parallel compilation is enabled.

This feature is available for Arria® series, Cyclone®, HardCopy III, HardCopy IV, MAX® II, MAX V (limited support), and Stratix® series devices.

For more information, refer to Processing Page (Options Dialog Box) in Quartus II Help.

For information about how to control the number of processors used during compilation for a specific project, refer to Compilation Process Settings Page (Settings Dialog Box) in Quartus II Help.

You can also set the number of processors available for Quartus II compilation using the following Tcl command in your script.

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

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

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

```
set_global_assignment -name NUM_PARALLEL_PROCESSORS ALL
```

Using multiple processors does not affect the quality of the fit. For a given Fitter seed on a specific design, the fit is exactly the same, regardless of whether the Quartus II software uses one processor or multiple processors. The only difference between compilations using a different number of processors is the compilation time.

**Using Incremental Compilation**

The incremental compilation feature can speed up design iteration time by up to 70% for small design changes, and helps you reach design timing closure more efficiently. You can speed up design iterations by recompiling only a particular design partition and merging results with previous compilation results from other partitions. You can also use physical synthesis optimization techniques for specific design partitions while leaving other parts of your design untouched to preserve performance.
If you are using a third-party synthesis tool, you can create separate atom netlist files for parts of your design that you already have synthesized and optimized so that you update only the parts of your design that change.

In the standard incremental compilation design flow, you can divide the top-level design into partitions, which the software can compile and optimize in the top-level Quartus II project. You can preserve fitting results and performance for completed partitions while other parts of your design are changing, which reduces the compilation time for each design iteration because the software does not synthesize or fit the unchanged partitions in your design.

The incremental compilation feature also facilitates team-based design flows by enabling designers to create and optimize design blocks independently, when necessary, and support third-party IP integration.

For information about the full incremental compilation flow in the Quartus II software, refer to the Quartus II Incremental Compilation for Hierarchical and Team-Based Design chapter in volume 1 of the Quartus II Handbook. For information about creating multiple netlist files in third-party tools for use with incremental compilation, refer to the appropriate chapter in Section IV. Synthesis in volume 1 of the Quartus II Handbook.

For additional information about incremental compilation, refer to About Incremental Compilation in Quartus II Help.

### Using the Smart Compilation Setting

Smart compilation can reduce compilation time by skipping unnecessary Compiler stages to recompile your design. This setting is especially useful when you perform multiple compilation iterations during the optimization phase of your design process. However, smart compilation uses more disk space. To turn on smart compilation, on the Assignments menu, click Settings. In the Category list, select Compilation Process Settings and turn on Use smart compilation.

Smart compilation skips unnecessary Compiler stages (such as Analysis and Synthesis). This feature is different from incremental compilation, which you can use to compile parts of your design while preserving results for unchanged parts.

### Using Rapid Recompile

The Rapid Recompile feature maximizes designer productivity when making small engineering change order (ECO)-style design changes after a full compilation, reducing compilation times by an average of 50%. Rapid Recompile also significantly improves designer productivity during timing closure by preserving critical timing during late design changes.

You can use the Rapid Recompile feature on its own or along with standard incremental flow for compatible nodes in your design. A compatible node is a node that you can match to a node from previous compilation results. Rapid Recompile allows the Quartus II software to reuse placement and routing resources of compatible nodes from previous results with a high degree of confidence.
If you enable the Rapid Recompile feature, you can view the compilation time reduction after a full compilation. Turn on the Rapid Recompile feature in later compilations to view further reductions. The Incremental Compilation Preservation Summary section in the Fitter Report provides details about the placement and routing preservation for your design.

The performance of Rapid Recompile is largely dependent on the nature of your design change. If the Quartus II software determines that full optimization is necessary for design performance, you may not see much compilation time reduction. For example, if the total time taken by the Fitter is dominated by the time taken for fitter preparation operations, using this feature may not save you a lot of compilation time. When you apply extensive global optimizations, a small user change may be required to obtain optimal performance. Be sure to select the right flow to achieve your end goals.

If you see the message Fitter has failed to locate previous placement information during the compilation of your design, Rapid Recompile does not provide any compile time reduction.

For more information about this feature, refer to Incremental Compilation Page (Settings Dialog Box) in Quartus II Help.

Reducing Synthesis Time and Synthesis Netlist Optimization Time

You can reduce synthesis time by reducing your use of netlist optimizations and by using incremental compilation (with Netlist Type set to Post-Synthesis) without affecting the Fitter time. For tips for reducing synthesis time when using third-party EDA synthesis tools, refer to your synthesis software’s documentation.

Settings to Reduce Synthesis Time and Synthesis Netlist Optimization Time

You can use Quartus II integrated synthesis to synthesize and optimize HDL designs, and you can use synthesis netlist optimizations to optimize netlists that were synthesized by third-party EDA software. When using Quartus II Integrated Synthesis, you can also enable specific Physical Synthesis Optimizations during Analysis and Synthesis. Using these netlist optimizations can cause the Analysis and Synthesis module to take much longer to run. Read the Analysis and Synthesis messages to find out how much time these optimizations take. The compilation time spent in Analysis and Synthesis is usually small compared to the compilation time spent in the Fitter.

If your design meets your performance requirements without synthesis netlist optimizations, turn off the optimizations to save time. If you require synthesis netlist optimizations to meet performance, you can optimize parts of your design hierarchy separately to reduce the overall time spent in Analysis and Synthesis.

Turn off settings that are not useful. In general, if you carry over compilation settings from a previous project, evaluate all settings and keep only those that you need.
Use Appropriate Coding Style to Reduce Synthesis Time

The method that you use to code your design in HDL can affect the synthesis time. For example, if you want to infer RAM blocks from your code, you must follow the guidelines for inferring RAMs. If not, the software implements those blocks as registers, and if you are trying to infer a large memory, the software uses a large amount of resources in the FPGA, causing routing congestion and increases compilation time drastically. If you see high routing utilizations in certain blocks, it is a good idea to review the code for such blocks.

For more information about coding guidelines, refer to the Recommended HDL Coding Styles chapter in volume 1 of the Quartus II Handbook.

Using Early Timing Estimation

The Quartus II software provides an Early Timing Estimation feature that estimates your design’s timing results before the software performs full placement and routing. On the Processing menu, point to Start, and click Start Early Timing Estimate to generate initial compilation results after you have run Analysis and Synthesis. When you want a quick estimate of a design’s performance before proceeding with further design or synthesis tasks, this command can save significant compilation time. Using this feature provides a timing estimate 2.5× faster (on average) than running a full compilation (8.5× faster in best case), although the fit is not fully optimized or routed. Therefore, the timing report is only an estimate. On average, the estimated delays are within 15% of the final timing results as achieved by a full compilation.

You can specify the type of delay estimates to use with Early Timing Estimation. On the Assignments menu, click Settings. In the Category list, select Compilation Process Settings, and select Early Timing Estimate. On the Early Timing Estimate page, the following options are available:

- The Realistic option, which is the default, generates delay estimates that are similar to the results of a full compilation.
- The Optimistic option uses delay estimates that are likely lower than those achieved by a full compilation, which results in an optimistic performance estimate.
- The Pessimistic option uses delay estimates that are likely higher than those achieved by a full compilation, which results in a pessimistic performance estimate.

All three options offer the same reduction in compilation time.

You can view the critical paths in your design by locating these paths in the Chip Planner from the TimeQuest Timing Report panel. Then, if necessary, you can add or modify floorplan constraints such as LogicLock regions, or make other changes to the design. You can then rerun the Early Timing Estimate to quickly assess the impact of any floorplan assignments or logic changes, enabling you to try different design variations and find the best solution.
Reducing Placement Time

The time required to place a design depends on two factors: the number of ways the logic in your design can be placed in the device and the settings that control how hard the Placer works to find a good placement. You can reduce the placement time in two ways:

- Change the settings for the placement algorithm
- Use incremental compilation to preserve the placement for parts of your design

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

Fitter Effort Setting

The highest Fitter effort setting, Standard Fit, requires the most runtime, but does not always yield a better result than using the default Auto Fit. For designs with very tight timing requirements, both Auto Fit and Standard Fit use the maximum effort during optimization. Altera recommends using Auto Fit for reducing compilation time. If you are certain that your design has only easy-to-meet timing constraints, you can select Fast Fit for an even greater runtime savings.

Placement Effort Multiplier Settings

You can control the amount of time the Fitter spends in placement by reducing one aspect of placement effort with the Placement Effort Multiplier option. On the Assignments menu, click Settings. Select Fitter Settings, and click More Settings. Under Existing Option Settings, select Placement Effort Multiplier. The default is 1.0. Legal values must be greater than 0 and can be non-integer values. Numbers between 0 and 1 can reduce fitting time, but also can reduce placement quality and design performance. Numbers higher than 1 increase placement time and placement quality, but can reduce routing time for designs with routing congestion. For example, a value of 4 increases placement time by approximately 2 to 4 times, but might result in better placement, which can result in reduced routing time.

Final Placement Optimization Levels

The Final Placement Optimization Level option specifies whether the Fitter performs final placement optimizations. You can set this option to Always, Never, or Automatically. Performing optimizations can improve register-to-register timing and fitting, but might require longer compilation times. You can use the default setting of Automatically with the Auto Fit Fitter Effort Level (also the default) to enable the Fitter to decide whether these optimizations should run based on the routability and timing requirements of your design.

Setting the Final Placement Optimization Level to Never often reduces your compilation time, but affects routability negatively and reduces timing performance.

To change the Final Placement Optimization Level, on the Assignments menu, click Settings. The Settings dialog box appears. From the Category list, select Fitter Settings, and then click the More Settings button. Select Final Placement Optimization Level, and then from the list, select the required setting.
Physical Synthesis Effort Settings

You can use the physical synthesis options to optimize your post-synthesis netlist and improve your timing performance. These options, which affect placement, can significantly increase compilation time.

If your design meets your performance requirements without physical synthesis options, turn them off to save time. You also can use the Physical synthesis effort setting on the Physical Synthesis Optimizations page under Compilation Process Settings in the Category list to reduce the amount of extra compilation time that these optimizations use. The Fast setting directs the Quartus II software to use a lower level of physical synthesis optimization that, compared to the Normal physical synthesis effort level, can cause a smaller increase in compilation time. However, the lower level of optimization can result in a smaller increase in design performance.

Limit to One Fitting Attempt

This option causes the software to quit after one fitting attempt, instead of repeating placement and routing with increased effort. For hard-to-fit designs, consider increasing the Placement Effort Multiplier setting and the Limit to One Fitting Attempt setting. Increasing the Placement Effort Multiplier and the Limit to One Fitting Attempt settings saves you time, because if your design is hard to fit and does not result in a valid fit, the compilation stops after the first attempt.

From the Assignments menu, select Settings. On the Fitter Settings page, turn on Limit to one fitting attempt.

For more details about this option, refer to “Limit to One Fitting Attempt” in the Area and Timing Optimization chapter in volume 2 of the Quartus II Handbook.

Preserving Placement with Incremental Compilation

Preserving information about previous placements can make future placements faster. The incremental compilation feature provides an easy-to-use methodology for preserving placement results. For more information, refer to “Using Incremental Compilation” on page 12–3.

Reducing Routing Time

The time required to route a design depends on three factors: the device architecture, the placement of your design in the device, and the connectivity between different parts of your design. The routing time is usually not a significant amount of the compilation time. If your design requires a long time to route, perform one or more of the following actions:

- Check for routing congestion
- Let the placer run longer to find a more routable placement
- Use incremental compilation to preserve routing information for parts of your design
Identifying Routing Congestion in the Chip Planner

To identify areas of routing congestion in your design, open the Chip Planner. On the Tools menu, click Chip Planner. To view the routing congestion in the Chip Planner, click the Layers icon located next to the Task menu. Under Background Color Map, select Routing Utilization. Even if average congestion is not very high, your design may have areas where congestion is very high in a specific type of routing. You can use the Chip Planner to identify areas of high congestion for specific interconnect types. You can change the connections in your design to reduce routing congestion. If the area with routing congestion is in a LogicLock region or between LogicLock regions, change or remove the LogicLock regions and recompile your design. If the routing time remains the same, the time is a characteristic of your design and the placement. If the routing time decreases, consider changing the size, location, or contents of LogicLock regions to reduce congestion and decrease routing time.

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

The Quartus II compilation messages contain information about average and peak interconnect usage. Peak interconnect usage over 75%, or average interconnect usage over 60%, could be an indication that it might be difficult to fit your design. Similarly, peak interconnect usage over 90%, or average interconnect usage over 75%, are likely to have increased chances of not getting a valid fit.

For information about identifying areas of congested routing using the Chip Planner, refer to the “Viewing Routing Congestion” subsection in the Analyzing and Optimizing the Design Floorplan chapter in volume 2 of the Quartus II Handbook.

Placement Effort Multiplier Setting

Some designs might be time consuming and difficult to route because the placement is not optimal. In such cases, you can increase the Placement Effort Multiplier to get a better placement. Increasing the Placement Effort Multiplier might increase the placement time, but sometimes it can reduce the routing time, and even overall compilation time.

Preserving Routing with Incremental Compilation

Preserving the previous routing results for part of your design can reduce future routing time. Incremental compilation provides an easy-to-use methodology that preserves placement and routing results. For more information, refer to “Using Incremental Compilation” on page 12–3 and the references listed in the section.

Reducing Static Timing Analysis Time

If you are performing timing-driven synthesis, the Quartus II software runs the TimeQuest analyzer during Analysis and Synthesis. The Quartus II Fitter also runs the TimeQuest analyzer during placement and routing. If there are incorrect constraints in the .sdc file, the Quartus II software may spend time processing constraints unnecessarily several times. If you do not specify false paths and multicyle paths in your design, the TimeQuest analyzer may spend time analyzing paths that are not relevant to your design. Also, if you redefine constraints in the .sdc...
files, the TimeQuest analyzer may spend additional time processing them. In the compilation messages, look for indications that Synopsis design constraints are redefined, and update the .sdc file to avoid this situation. Also, ensure that you provide the correct timing constraints to your design, because the software cannot assume design intent, such as which paths to consider as false paths or multicycle paths. When you specify these assignments correctly, the TimeQuest analyzer skips analysis for those paths, and the Fitter does not spend additional time optimizing those paths.

**Setting Process Priority**

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

For more information about setting process priority, refer to *Processing Page (Options Dialog Box)* in Quartus II Help.

**Conclusion**

The Quartus II software provides many features to reduce compilation time and achieve optimal results. Using the recommended techniques described in this chapter can help you reduce compilation time.

**Document Revision History**

Table 12–1 shows the revision history for this chapter.

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Changes</th>
</tr>
</thead>
</table>
| May 2011    | 11.0.0  | ■ Updated “Using Parallel Compilation with Multiple Processors” on page 12–2.  
■ Updated “Identifying Routing Congestion in the Chip Planner” on page 12–9.  
■ General editorial changes throughout the chapter. |
| December 2010| 10.1.0  | ■ Template update.  
■ Added details about peak and average interconnect usage.  
■ Added new section “Reducing Static Timing Analysis Time” on page 12–9.  
■ Minor changes throughout chapter. |
| July 2010   | 10.0.0  | Initial release. |

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

Take an [online survey](#) to provide feedback about this handbook chapter.
This chapter describes techniques to reduce resource usage and improve timing performance when designing for Altera® devices.

Good optimization techniques are essential for achieving the best results when designing for programmable logic devices (PLDs). The optimization features available in the Quartus® II software allow you to meet design requirements by applying these techniques at multiple points in the design process.

This chapter also explains how and when to use some of the features described in other chapters of the Quartus II Handbook.

This chapter includes the following topics:

- “Optimizing Your Design”
- “Design Analysis” on page 13–9
- “Resource Utilization Optimization Techniques (LUT-Based Devices)” on page 13–15
- “Timing Optimization Techniques (LUT-Based Devices)” on page 13–26
- “Resource Utilization Optimization Techniques (Macrocell-Based CPLDs)” on page 13–42
- “Timing Optimization Techniques (Macrocell-Based CPLDs)” on page 13–48
- “Scripting Support” on page 13–53

The application of these techniques varies from design to design. Applying each technique does not always improve results. Settings and options in the Quartus II software have default values that generally 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.

You can use the optimization flow described in this chapter to explore various compiler settings and determine the techniques that provide the best results.

**Optimizing Your Design**

The first stage in the optimization process is to perform an initial compilation of your design. “Initial Compilation: Required Settings” on page 13–2 provides guidelines for some of the settings and assignments that are recommended for your initial compilation. “Initial Compilation: Optional Fitter Settings” on page 13–5 describes settings that you might turn on based on your design requirements. “Design Analysis” on page 13–9 explains how to analyze the compilation results.

You can use incremental compilation in the optimization process. Incremental compilation can preserve timing to aid in timing closure, as well as compilation time reduction; however, it can cause a slight increase in resource utilization.
For more details about Quartus II incremental compilation flow, refer to the *Quartus II Incremental Compilation for Hierarchical and Team-Based Design* chapter in volume 1 of the *Quartus II Handbook*.

To view information about timing analysis results, refer to *Viewing Timing Analysis Results (TimeQuest Timing Analyzer)* in Quartus II Help.

After you have analyzed the results from an initial compilation, perform the optimization stages in the recommended order, as described in this chapter.

For LUT-based devices (FPGAs, MAX® II series devices), perform optimizations in the following order:

1. If your design does not fit, refer to “Resource Utilization Optimization Techniques (LUT-Based Devices)” on page 13–15 before trying to optimize I/O timing or register-to-register timing.

2. If your design does not meet the required I/O timing performance, refer to “I/O Timing Optimization Techniques (LUT-Based Devices)” on page 13–55 before trying to optimize register-to-register timing.

3. If your design does not meet the required slack on any of the clock domains in the design, refer to “Register-to-Register Timing Optimization Techniques (LUT-Based Devices)” on page 13–55.

For macrocell-based devices (MAX 7000 and MAX 3000 CPLDs), perform optimizations in the following order:

1. If your design does not fit, refer to “Resource Utilization Optimization Techniques (Macrocell-Based CPLDs)” on page 13–42 before trying to optimize I/O timing or register-to-register timing.

2. If your timing performance requirements are not met, refer to “Timing Optimization Techniques (Macrocell-Based CPLDs)” on page 13–48.

For device-independent techniques to reduce compilation time, refer to the “Compilation-Time Optimization Techniques” section in the *Reducing Compilation Time* chapter in volume 2 of the *Quartus II Handbook*.

You can use these techniques in the GUI or with Tcl commands. For more information about scripting techniques, refer to “Scripting Support” on page 13–53.

### Initial Compilation: Required Settings

This section describes the basic assignments and settings for your initial compilation. Check the following settings before compiling the design in the Quartus II software. Significantly varied compilation results can occur depending on the assignments you set.

Verify the following settings:

- “Device Settings” on page 13–3
- “I/O Assignments”
- “Timing Requirement Settings”
- “Device Migration Settings” on page 13–5
“Partitions and Floorplan Assignments for Incremental Compilation” on page 13–5

Device Settings
Specific device assignments determine the timing model that the Quartus II software uses during compilation. Choose the correct speed grade to obtain accurate results and the best optimization. The device size and the package determine the device pin-out and the number of resources available in the device.

I/O Assignments
The I/O standards and drive strengths specified for a design affect I/O timing. Specify I/O assignments so that the Quartus II software uses accurate I/O timing delays in timing analysis and Fitter optimizations.

The Quartus II software can select pin locations automatically. If your pin locations are not fixed due to PCB layout requirements, leave pin locations unconstrained. If your pin locations are already fixed, make pin assignments to constrain the compilation appropriately. “Resource Utilization Optimization Techniques (Macrocell-Based CPLDs)” on page 13–42 includes recommendations for making pin assignments that can have a large effect on your results in smaller macrocell-based architectures.

Use the Assignment Editor and Pin Planner to assign I/O standards and pin locations.

For more information about I/O standards and pin constraints, refer to the appropriate device handbook. For information about planning and checking I/O assignments, refer to the I/O Management chapter in volume 2 of the Quartus II Handbook.

For information about using the Assignment Editor, refer to About the Assignment Editor in Quartus II Help.

Timing Requirement Settings
You must use comprehensive timing requirement settings to achieve the best results for the following reasons:

- Correct timing assignments allow the software to work hardest to optimize the performance of the timing-critical parts of the design and make trade-offs for performance. This optimization can also save area or power utilization in non-critical parts of the design.

- The Quartus II software performs physical synthesis optimizations based on timing requirements (refer to “Physical Synthesis Optimizations” on page 13–35 for more information).

- Depending on the Fitter Effort setting, the Quartus II Fitter can reduce runtime considerably if your timing requirements are being met.

For a description of the different effort levels, refer to “Fitter Effort Setting” on page 13–7.
Use your real requirements to get the best results. If you apply more demanding timing requirements than you actually need, increased resource usage, higher power utilization, increased compilation time, or all of these may result.

The Quartus II TimeQuest Timing Analyzer checks your design against the timing constraints. The Compilation Report and timing analysis reporting commands show whether timing requirements are met and provide detailed timing information about paths that violate timing requirements.

To create timing constraints for the TimeQuest analyzer, create a Synopsys Design Constraints File (.sdc). You can also enter constraints in the TimeQuest GUI. Use the write_sdc command, or, on the Constraints menu in the TimeQuest analyzer, click Write SDC File to write your constraints to an .sdc. You can add an .sdc to your project on the Quartus II Settings page under Timing Analysis Settings.

If you already have an .sdc in your project, using the write_sdc command from the command line or using the Write SDC File option from the TimeQuest GUI enables you to create a new .sdc, combining the constraints from your current .sdc and any new constraints added through the GUI or command window, or overwriting the existing .sdc with your newly applied constraints.

Ensure that every clock signal has an accurate clock setting constraint. If clocks arrive from a common oscillator, they can be considered related. Ensure that all related or derived clocks are set up correctly in the constraints. All I/O pins that require I/O timing optimization must be constrained. Specify both minimum and maximum timing constraints as applicable. If there is more than one clock or there are different I/O requirements for different pins, make multiple clock settings and individual I/O assignments instead of using a global constraint.

Make any complex timing assignments required in the design, including false path and multicycle path assignments. Common situations for these types of assignments include reset or static control signals, cases in which it is not important how long it takes a signal to reach a destination, and paths that can operate in more than one clock cycle. These assignments allow the Quartus II software to make appropriate trade-offs between timing paths and can enable the Compiler to improve timing performance in other parts of the design.

For more information about timing assignments and timing analysis, refer to The Quartus II TimeQuest Timing Analyzer chapter in volume 3 of the Quartus II Handbook and the Quartus II TimeQuest Timing Analyzer Cookbook.

To ensure that constraints or assignments have been applied to all design nodes, you can report all unconstrained paths in your design.

While using the Quartus II TimeQuest analyzer, you can report all the unconstrained paths in your design with the Report Unconstrained Paths command in the Task pane or the report_ucp Tcl command.
Device Migration Settings

If you anticipate a change to the target device later in the design cycle, either because of changes in the design or other considerations, plan for it at the beginning of your design cycle. Whenever you select a target device, you can also list any other compatible devices you can migrate to by clicking on the Migration Devices button in the Device dialog box. If you plan to move your design to a HardCopy® device, make sure to select the device from the HardCopy list under Companion device in the Device dialog box.

Selecting the migration device and companion device early in the design cycle helps to minimize changes to the design at a later stage.

Partitions and Floorplan Assignments for Incremental Compilation

The Quartus II incremental compilation feature enables hierarchical and team-based design flows in which you can compile parts of your design while other parts of the design remain unchanged, or import parts of your design from separate Quartus II projects.

Using incremental compilation for your design with good design partitioning methodology can help to achieve timing closure. Creating design partitions on some of the major blocks in your design, and assigning them to not too restrictive LogicLock™ regions generally reduces Fitter time, and improves the quality and repeatability of results. Using incremental compilation can help you achieve timing closure block by block, and preserve the timing performance between iterations, which helps achieve timing closure for the entire design. Using incremental compilation may also help reduce compilation times.

For more information, refer to the “Incremental Compilation” section in the Reducing Compilation Time chapter in volume 2 of the Quartus II Handbook.

If you want to take advantage of incremental compilation for a team-based design flow to reduce your compilation times, or to improve the timing performance of your design during iterative compilation runs, make meaningful design partitions and create a floorplan for your design partitions.

If you plan to use incremental compilation, you must create a floorplan for your design. If you are not using incremental compilation, this step is optional.

For guidelines about how to create partition and floorplan assignments for your design, refer to the Best Practices for Incremental Compilation Partitions and Floorplan Assignments chapter in volume 1 of the Quartus II Handbook.

Initial Compilation: Optional Fitter Settings

This section describes optional Fitter settings that can help optimize your design. You can selectively set all the optional settings that help to improve performance. These settings vary between designs and there is no standard set that applies to all designs. Significantly different compilation results can occur depending on the assignments you have set.
The following settings are optional:

- “Optimize Hold Timing”
- “Optimize Multi-Corner Timing” on page 13–7
- “Fitter Effort Setting” on page 13–7
- “Limit to One Fitting Attempt” on page 13–9
- “Design Assistant” on page 13–9

To turn on these settings, follow these steps:

1. On the Assignments menu, click **Settings**.
2. In the **Category** list, select **Fitter Settings**. The **Fitter Settings** page appears.
3. Turn on the appropriate options.

**Optimize Hold Timing**

The **Optimize Hold Timing** option directs the Quartus II software to optimize minimum delay timing constraints. This option is available for all Altera device families except MAX 3000 and MAX 7000 series devices. 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 guarantee that the minimum delay requirements are satisfied. 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 device input pins to 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 in Figure 13–1, where a derived clock generated with logic causes a hold time problem on another register. However, if your design has internal hold time violations between registers, correct the problems by making changes to your design, such as using a clock enable signal instead of a derived or gated clock.

**Figure 13–1. Optimize Hold Timing Option Fixing an Internal Hold Time Violation**

For design practices that can help eliminate internal hold time violations, refer to the **Recommended Design Practices** chapter in volume 1 of the **Quartus II Handbook**.
Optimize Multi-Corner Timing

Historically, FPGA timing analysis has been performed using only delays from the slow corner timing model. However, due to process variation and changes in the 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 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 compilation results. Therefore, the Quartus II software provides the 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.

By default, the Fitter optimizes constraints using only the slow corner timing model. You can turn on the Optimize multi-corner timing option to instruct the Fitter to also optimize constraints considering all available timing corners, at the cost of a slight increase in runtime. By optimizing for all timing corners, you can create a design implementation that is more robust across process, temperature, and voltage variations. While optimizing for multi-corner timing, the Fitter chooses one of the two slow corners that is known to have more critical timing (depending on the chosen device), along with the fast corner. This option is available only for Arria, Cyclone, HardCopy, MAX II, MAX V, and Stratix series devices.

Using the different timing models can be important to account for process, voltage, and temperature variations for each device. Turning this option on increases compilation time by approximately 10%.

For designs with external memory interfaces such as DDR and QDR, Altera recommends that you turn on the Optimize multi-corner timing setting.

Fitter Effort Setting

Fitter effort refers to the amount of effort the Quartus II software uses to fit your design. To set the Fitter effort, on the Assignments menu, click Settings. In the Category list, select Fitter Settings. The Fitter effort settings are Auto Fit, Standard Fit, and Fast Fit. The default setting depends on the device family specified. Auto Fit is the default Fitter effort setting for all devices for which this option is available.

Auto Fit

The Auto Fit option (available for Arria, Cyclone, HardCopy, MAX II, MAX V, and Stratix series devices) focuses the full Fitter effort only on those aspects of the design that require further optimization. Auto Fit can significantly reduce compilation time relative to Standard Fit if your design has easy-to-meet timing requirements, low routing resource utilization, or both. However, those designs that require full optimization generally receive the same effort as is achieved by selecting Standard Fit.

If you want the Fitter to attempt to exceed the timing requirements by a certain margin instead of simply meeting them, specify a minimum slack in the Desired worst case slack box.
Specifying a minimum slack does not guarantee that the Fitter achieves the slack requirement; it only guarantees that the Fitter applies full optimization unless the target slack is exceeded.

In some designs with multiple clocks, it might be possible to improve the timing performance on one clock domain while reducing the performance on other clock domains by over-constraining the most important clock. If you use this technique, perform a sweep over multiple seeds to ensure that any performance improvements that you see are real gains. For more information, refer to “Fitter Seed” on page 13–39.

Over-constraining the clock for which you require maximum slack, while using the Auto Fit option, increases the chances that the Fitter is able to meet this requirement.

The Auto Fit option also causes the Quartus II Fitter to optimize for shorter compilation times instead of maximum possible performance if the design includes easy to achieve timing requirements.

If your design has aggressive timing requirements or is hard to route, the placement does not stop early and the compilation time is the same as using the Standard Fit option.

It is possible for the Auto Fit option to increase routing utilization. This can lead to an increase in dynamic power when compared to using the Standard Fit option, unless the Extra effort option in the PowerPlay power optimization list is also enabled. When you turn on Extra effort, Auto Fit continues to optimize for reduction of routing usage even after meeting the register-to-register requirement, and there is no adverse effect on the dynamic power consumption relative to using Standard Fit. If dynamic power consumption is a concern, select Extra effort in both the Analysis & Synthesis Settings and the Fitter Settings pages.

For more details, refer to the “Power Driven Compilation” section in the Power Optimization chapter in volume 2 of the Quartus II Handbook.

**Standard Fit**

Use the Standard Fit option to exceed specified timing requirements and achieve the best possible timing results and lowest routing resource utilization for your design. The Standard Fit setting usually increases compilation time relative to Auto Fit, because it applies full optimization, regardless of the design requirement. In designs with no timing assignments, on average, using the Standard Fit option results in a \( f_{\text{MAX}} \) about 10% higher than that achieved using the Auto Fit option. In designs where timing requirements can be easily met, using the Standard Fit option can result in considerably longer compilation times than using the Auto Fit option.

**Fast Fit**

The Fast Fit option reduces the amount of optimization effort for each algorithm employed during fitting. This option reduces the compilation time by about 50%, resulting in a fit that has, on average, 10% lower \( f_{\text{MAX}} \) than that achieved using the Standard Fit setting.
Limit to One Fitting Attempt

A design might fail to fit for several reasons, such as logic overuse or illegal assignments. For most failures, the Quartus II software informs you of the problem. However, if the design uses too much routing, the Quartus II software makes up to two additional attempts to fit your design, increasing the Placement Effort Multiplier each time. Each of these fit attempts takes significantly longer than the previous attempt.

For large designs, you might not want to wait for all three fitting attempts to be completed. To have the Quartus II software issue an error message after the first failed attempt, turn on Limit to one fitting attempt on the Fitter Settings page.

For instructions about how to lower the design’s routing utilization, so your design can be made to fit into the target device if it fails to fit due to the lack of routing resources, refer to “Routing” on page 13–23.

Design Assistant

You can run the Design Assistant to analyze the post-fitting results of your design during a full compilation. The Design Assistant checks rules related to gated clocks, reset signals, asynchronous design practices, and signal race conditions. This is especially useful during the early stages of your design, so that you can work on any areas of concern in your design before proceeding with design optimization.

For more information about the Design Assistant, refer to About the Design Assistant and Analyzing Designs with the Design Assistant in Quartus II Help.

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.

Error and Warning Messages

After compiling your design, evaluate all error and warning messages to see if any design or setting changes are required. If changes are required, make these changes and recompile the design before proceeding with design optimization.

To suppress messages that you have already evaluated and do not want to see again, right-click on the message in the Messages window and click Suppress.

For more information about message suppression, refer to the “Message Suppression” section in the Managing Quartus II Projects chapter in volume 2 of the Quartus II Handbook.

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:

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

If any constraints were ignored, analyze why they were ignored. If necessary, correct the constraints and recompile the 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 in volume 3 of the Quartus II Handbook.

**Resource Utilization**

Determining device utilization is important regardless of whether a successful fit is achieved. If your compilation results in a no-fit error, resource utilization information is important for analyzing the fitting problems in your design. If your fitting is successful, review the resource utilization information to determine whether the future addition of extra logic or other design changes might introduce fitting difficulties. Also, review the resource utilization information to determine if it is impacting timing performance.

To determine resource usage, refer to the Flow Summary section of the Compilation Report. This section reports how many resources are used, including pins, memory bits, digital signal processing, and phase-locked loops (PLLs). The Flow Summary indicates whether the design exceeds the available device resources. More detailed information is available by viewing the reports under Resource Section in the Fitter section of the Compilation Report.

The Flow Summary shows the overall logic utilization, and also individual utilization for combinational ALUTS, memory ALUTs, and registers. The overall logic utilization could be higher than the numbers for combinational logic or register utilization numbers may indicate. This is because the Fitter uses adaptive look-up tables (ALUTs) in different ALMs—even when the logic can be placed within one ALM—to achieve the best timing and routing results. The Fitter can spread logic throughout the device, which may lead to higher overall utilization.

As the device fills up, the Fitter automatically searches for logic functions with common inputs to place in one ALM. The number of partnered ALUTs and packed registers also increases. Therefore, a design that has high overall utilization might still have space for extra logic if logic and registers can be packed together more aggressively.

The reports under Resource Section in the Fitter section of the Compilation Report provide more detailed resource information. The Fitter Resource Usage Summary report breaks down the logic utilization information, indicates the number of fully and partially used ALMs, and provides other resource information, including the number of bits in each type of memory block. This panel also contains a summary of the usage of global clocks, PLLs, DSP blocks, and other device-specific resources.
You can also view reports describing some of the optimizations that occurred during compilation. For example, if you are using Quartus II integrated synthesis, the reports in the Optimization Results folder in the Analysis & Synthesis section include information about registers that were removed during synthesis. Use this report to estimate device resource utilization for a partial design to ensure that registers were not removed due to missing connections with other parts of the design.

If a specific resource usage is reported as less than 100% and a successful fit cannot be achieved, either there are not enough routing resources, or some assignments are illegal. In either case, a message appears in the Processing tab of the Messages window describing the problem.

If the Fitter finishes unsuccessfully and runs much faster than on similar designs, a resource might be over-utilized or there might be an illegal assignment. If the Quartus II software seems to run for an excessively long time compared to runs on similar designs, a legal placement or route probably cannot be found. In the Compilation Report, look for errors and warnings that indicate these types of problems.

For more information about how to get a quick error message on hard-to-fit designs, refer to “Limit to One Fitting Attempt” on page 13–9.

You can use the Chip Planner to find areas of the device that have routing congestion on specific types of routing resources. If you find areas with very high congestion, analyze the cause of the congestion. Issues such as high fan-out nets not using global resources, an improperly chosen optimization goal (speed versus area), very restrictive floorplan assignments, or the coding style can cause routing congestion. After you identify the cause, modify the source or settings to reduce routing congestion.

For information about how to view routing congestion, refer to Displaying Resources and Information in Quartus II Help.

For details about using the Chip Planner tool, refer to the Analyzing and Optimizing the Design Floorplan chapter in volume 2 of the Quartus II Handbook and About the Chip Planner in Quartus II Help.

**I/O Timing (Including t\textsubscript{PD})**

TimeQuest analyzer supports the Synopsys Design Constraints (SDC) format for constraining your design. When using the TimeQuest analyzer for timing analysis, use the \texttt{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 \texttt{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 \texttt{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 in volume 3 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 13–14
- “Tips for Analyzing Failing Clock Paths that Cross Clock Domains” on page 13–14

### Timing Analysis with the TimeQuest Timing Analyzer

If you are using the TimeQuest analyzer, analyze all valid register-to-register paths by using appropriate constraints. Use the `report_timing` command to generate the required timing reports for any register-to-register path. 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 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 (Figure 13–2). To locate a selected path in the Chip Planner or the Technology Map Viewer by using the shortcut menu, right-click on a path, point to *Locate*, and click *Locate in Chip Planner*. The Chip Planner appears with the path highlighted. Similarly, if you know that a path is not a valid path, you can set it to be a false path using the shortcut menu.

To see the path details of any selected path, click on the *Data Path* tab in the path details pane. This displays the details of the Data Arrival Path, as well as the Data Required Path. For a graphical view of the information, click on the *Waveform* tab.
You can locate critical paths in the Chip Planner from the TimeQuest timing analysis report panel.

Figure 13–2. TimeQuest Analyzer GUI

For more information about how timing analysis results are calculated, refer to the Quartus II TimeQuest Timing Analyzer chapter in volume 3 of the Quartus II Handbook.

You also can see the logic in a particular path by locating the logic in the RTL Viewer or Technology Map Viewer. These viewers allow you to see a gate-level or technology-mapped representation of your design netlist. To locate a timing path in one of the viewers, right-click on a path in the report, point to Locate, and click Locate in RTL Viewer or Locate in Technology Map Viewer. When you locate a timing path in the Technology Map Viewer, the annotated schematic displays the same delay information that is shown when you use the List Paths command.

For more information about netlist viewers, refer to the Analyzing Designs with Quartus II Netlist Viewers chapter in volume 1 of the Quartus II Handbook.
**Tips for Analyzing Failing Paths**

When you are analyzing clock path failures, examine reports and waveforms to determine if the correct constraints are being applied, and add multicycle or false paths as appropriate.

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.

Check for particular nodes that appear in many failing paths. 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 be helpful 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 *From* 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 between two clock domains. This is the case if the *From Clock* and *To Clock* in the timing analysis report are different. 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. To analyze these paths in more detail, right-click on the entry in the report and click *List Paths*.

Expand the *List Paths* entry in the Messages window and analyze the largest register-to-register requirement. Evaluate the setup relationship between the source and destination (launch edge and latch edge) to determine if that is reducing the available setup time. 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.

Check to see 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, 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 in volume 3 of the *Quartus II Handbook*.

Evaluate the clock skew between the source clock and the destination clock to determine if that is reducing the available setup time. You can check the shortest and longest clock path reports to see what is causing the clock skew. Avoid using combinational logic in clock paths because it contributes to clock skew. Differences in the logic or in its routing between the source and destination can cause clock skew problems and result in warnings during compilation.

**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 appropriate signals have been placed on global routing resources. In the Compilation Report, open the Fitter report and click the **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 clock 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.

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

After design analysis, the next stage of design optimization is to improve resource utilization. Complete this stage before proceeding to I/O timing optimization or register-to-register timing optimization. Ensure that you have already set the basic constraints described in “Initial Compilation: Required Settings” on page 13–2 before proceeding with the resource utilization optimizations discussed in this section. If a design does not fit into a specified device, use the techniques in this section to achieve a successful fit. After you optimize resource utilization and your design fits in the desired target device, optimize I/O timing as described in “I/O Timing Optimization Techniques (LUT-Based Devices)” on page 13–55. These tips are valid for all FPGA families and the MAX II family of CPLDs.
Using the Resource Optimization Advisor

The Resource Optimization Advisor provides guidance in determining settings that optimize the resource usage. To run the Resource Optimization Advisor, on the Tools menu, point to Advisors, and click Resource Optimization Advisor.

The Resource Optimization Advisor provides step-by-step advice about how to optimize the resource usage (logic element, memory block, DSP block, I/O, and routing) of your design. Some of the recommendations in these categories might conflict with each other. Altera recommends evaluating the options and choosing the settings that best suit your requirements.

Resolving Resource Utilization Issues Summary

Resource utilization issues can be divided into the following three categories:

- Issues relating to I/O pin utilization or placement, including dedicated I/O blocks such as PLLs or LVDS transceivers (refer to “I/O Pin Utilization or Placement”).
- Issues relating to logic utilization or placement, including logic cells containing registers and look-up tables as well as dedicated logic, such as memory blocks and DSP blocks (refer to “Logic Utilization or Placement” on page 13–17).
- Issues relating to routing (refer to “Routing” on page 13–23).

I/O Pin Utilization or Placement

Use the suggestions in the following sections to help you resolve I/O resource problems.

Use I/O Assignment Analysis

On the Processing menu, point to Start and click Start I/O Assignment Analysis to help with pin placement. The Start I/O Assignment Analysis command allows you to check your I/O assignments early in the design process. You can use this command to check the legality of pin assignments before, during, or after compilation of your design. If design files are available, you can use this command to accomplish more thorough legality checks on your design’s I/O pins and surrounding logic. These checks include proper reference voltage pin usage, valid pin location assignments, and acceptable mixed I/O standards.

Common issues with I/O placement relate to the fact that differential standards have specific pin pairings, and certain I/O standards might be supported only on certain I/O banks.

If your compilation or I/O assignment analysis results in specific errors relating to I/O pins, follow the recommendations in the error message. Right-click on the message in the Messages window and click Help to open the Quartus II Help topic for this message.

Modify Pin Assignments or Choose a Larger Package

If a design that has pin assignments fails to fit, compile the design without the pin assignments to determine whether a fit is possible for the design in the specified device and package. You can use this approach if a Quartus II error message indicates fitting problems due to pin assignments.
If the design fits when all pin assignments are ignored or when several pin assignments are ignored or moved, you might have to modify the pin assignments for the design or select a larger package.

If the design fails to fit because insufficient I/Os are available, a successful fit can often be obtained by using a larger device package (which can be the same device density) that has more available user I/O pins.

For more information about I/O assignment analysis, refer to the I/O Management chapter in volume 2 of the Quartus II Handbook.

Logic Utilization or Placement

Use the suggestions in the following subsections to help you resolve logic resource problems, including logic cells containing registers and lookup tables (LUTs), as well as dedicated logic such as memory blocks and DSP blocks.

Optimize Source Code

If your design does not fit because of logic utilization, evaluate if you can, and modify the design at the source to achieve the desired results. You can often improve logic significantly by making design-specific changes to your source code. This is typically the most effective technique for improving the quality of your results.

If your design does not fit into available LEs or ALMs, but you have unused memory or DSP blocks, check to see if you have code blocks in your design that describe memory or DSP functions that are not being inferred and placed in dedicated logic. You might be able to modify your source code to allow these functions to be placed into dedicated memory or DSP resources in the target device.

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 for the State Machine report under Analysis & Synthesis in the Compilation Report. This report provides details, including the state encoding for each state machine that was recognized during compilation. If your state machine is not being 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 and DSP functions, refer to the “Instantiating Altera Megafuncons” and the “Inferring Multiplier and DSP Functions from HDL Code” sections of the Recommended HDL Coding Styles chapter in volume 1 of the Quartus II Handbook. For guidelines and sample HDL code for state machines, refer to the “General Coding Guidelines” section of the Recommended HDL Coding Styles chapter in volume 1 of the Quartus II Handbook.

For additional HDL coding examples, refer to AN 584: Timing Closure Methodology for Advanced FPGAs.
Optimize Synthesis for Area, Not Speed

If your design fails to fit because it uses too much logic, resynthesize the design to improve the area utilization. First, ensure that you have set your device and timing constraints correctly in your synthesis tool. Particularly when area utilization of the design is a concern, ensure that you do not over-constrain the timing requirements for the design. Synthesis tools generally try to meet the specified requirements, which can result in higher device resource usage if the constraints are too aggressive.

If resource utilization is an important concern, some synthesis tools offer an easy way to optimize for area instead of speed. If you are using Quartus II integrated synthesis, select Balanced or Area for the Optimization Technique. You can also specify this logic option for specific modules in your design with the Assignment Editor in cases where you want to reduce area using the Area setting (potentially at the expense of register-to-register timing performance) while leaving the default Optimization Technique setting at Balanced (for the best trade-off between area and speed for certain device families) or Speed. You can also use the Speed Optimization Technique for Clock Domains logic option to specify that all combinational logic in or between the specified clock domain(s) is optimized for speed.

In some synthesis tools, not specifying an fMAX requirement can result in less resource utilization.

In the Quartus II software, the Balanced setting typically produces utilization results that are very similar to those produced by the Area setting, with better performance results. The Area setting can give better results in some cases.

For more information about setting timing requirements and synthesis options in Quartus II integrated synthesis and other synthesis tools, refer to the appropriate chapter in Section III. Synthesis in volume 1 of the Quartus II Handbook, or your synthesis software's documentation.

The Quartus II software provides additional attributes and options that can help improve the quality of your synthesis results.

Restructure Multiplexers

Multiplexers form a large portion of the logic utilization in many FPGA designs. By optimizing your multiplexed logic, you can achieve a more efficient implementation in your Altera device.

For more information about this option, refer to Restructure Multiplexers logic option in Quartus II Help.

For design guidelines to achieve optimal resource utilization for multiplexer designs, refer to the Recommended HDL Coding Styles chapter in volume 1 of the Quartus II Handbook.

Perform WYSIWYG Primitive Resynthesis with Balanced or Area Setting

For information about this logic option, refer to Perform WYSIWYG Primitive Resynthesis logic option in Quartus II Help.
The **Balanced** setting typically produces utilization results that are very similar to the **Area** setting with better performance results. The **Area** setting can give better results in some cases. Performing WYSIWYG resynthesis for area in this way typically reduces register-to-register timing performance.

**Use Register Packing**

The **Auto Packed Registers** option implements the functions of two cells into one logic cell by combining the register of one cell in which only the register is used with the LUT of another cell in which only the LUT is used. Figure 13–3 shows register packing and the gain of one logic cell in the design.

![Register Packing](image1)

Figure 13–3. Register Packing

Registers can also be packed into DSP blocks (Figure 13–4).

![Register Packing in DSP Blocks](image2)

Figure 13–4. Register Packing in DSP Blocks

The following list shows the most common cases in which register packing helps to optimize a design:

- A LUT can be implemented in the same cell as an unrelated register with a single data input
- A LUT can be implemented in the same cell as the register that is fed by the LUT
- A LUT can be implemented in the same cell as the register that feeds the LUT
A register can be packed into a RAM block
A register can be packed into a DSP block
A register can be packed into an I/O Element (IOE)

For more information, refer to *Auto Packed Registers* logic option in Quartus Help.

**Remove Fitter Constraints**

A design with conflicting constraints or constraints that are difficult to meet may not fit in the targeted device. This can occur when the location or LogicLock assignments are too strict and not enough routing resources are available on the device.

In this case, use the *Routing Congestion* task in the Chip Planner to locate routing problems in the floorplan, then remove any location or LogicLock region assignments in that area. If your design still does not fit, the design is over-constrained. To correct the problem, remove all location and LogicLock assignments and run successive compilations, incrementally constraining the design before each compilation. You can delete specific location assignments in the Assignment Editor or the Chip Planner. To remove LogicLock assignments in the Chip Planner, in the LogicLock Regions Window, or on the Assignments menu, click *Remove Assignments*. Turn on the assignment categories you want to remove from the design in the *Available assignment categories* list.

For more information about the *Routing Congestion* task in the Chip Planner, refer to *Analyzing and Optimizing the Design Floorplan* in volume 2 of the *Quartus II Handbook*.

**Change State Machine Encoding**

State machines can be encoded using various techniques. Using binary or gray code encoding typically results in fewer state registers than one-hot encoding, which requires one register for every state bit. If your design contains state machines, changing the state machine encoding to one that uses the minimal number of registers may reduce resource utilization. The effect of state machine encoding varies depending on the way your design is structured.

If your design does not manually encode the state bits, you can specify the state machine encoding in your synthesis tool. When using Quartus II integrated synthesis, turn on the *Minimal Bits* setting for the *State Machine Processing* option.

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

You can also specify this logic option for specific modules or state machines in your design with the Assignment Editor.

You can also use the following Tcl command in scripts to modify the state machine encoding.

```
set_global_assignment -name state_machine_processing <value>
```

In this case, `<value>` can be AUTO, ONE-HOT, MINIMAL BITS, or USER-ENCODE.
Flatten the Hierarchy During Synthesis

Synthesis tools typically provide the option of preserving hierarchical boundaries, which can be useful for verification or other purposes. However, optimizing across hierarchical boundaries allows the synthesis tool to perform the most logic minimization, which can reduce area. Therefore, to achieve the best results, flatten your design hierarchy whenever possible.

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 in volume 1 of the Quartus II Handbook.

Retarget Memory Blocks

If your design fails to fit because it runs out of device memory resources, your design may require a certain type of memory the device does not have. For example, a design that requires two M-RAM blocks cannot be targeted to a Stratix EP1S10 device, which has only one M-RAM block. You might be able to obtain a fit by building one of the memories with a different size memory block, such as an M4K memory block.

If the memory block was created with the MegaWizard™ Plug-In Manager, open the MegaWizard Plug-In Manager and edit the RAM block type so it targets a new memory block size.

ROM and RAM memory blocks can also be inferred from your HDL code, and your synthesis software can place large shift registers into memory blocks by inferring the ALTSHIFT_TAPS megafuncton. This inference can be turned off in your synthesis tool to cause the memory or shift registers to be placed in logic instead of in memory blocks. Also, for improved timing performance, you can turn this inference off to prevent registers from being moved into RAM.

For more information, refer to Auto RAM Replacement logic option, Auto ROM Replacement logic option, and Auto Shift Register Replacement logic option in Quartus II Help.

Depending on your synthesis tool, you can also set the RAM block type for inferred memory blocks. In Quartus II integrated synthesis, set the ramstyle attribute to the desired memory type for the inferred RAM blocks, or set the option to logic, to implement the memory block in standard logic instead of a memory block.

Consider the resource utilization by hierarchy in the report file, and determine whether there is an unusually high register count in any of the modules. Some coding styles can prevent the Quartus II software from inferring RAM blocks from the source code because of their architectural implementation, and forces the software to implement the logic in flipflops. As an example, a function such as an asynchronous reset on a register bank might make it incompatible with the RAM blocks in the device architecture, so that the register bank is implemented in flipflops. It is often possible to move a large register bank into RAM by slight modification of associated logic.
For more information about memory inference control in other synthesis tools, refer to the appropriate chapter in *Section III. Synthesis* in volume 1 of the *Quartus II Handbook*, or your synthesis software’s documentation. For more information about coding styles and HDL examples that ensure memory inference, refer to the *Recommended HDL Coding Styles* chapter in volume 1 of the *Quartus II Handbook*.

**Use Physical Synthesis Options to Reduce Area**

The physical synthesis options for fitting can help you decrease the resource usage. When you enable these settings for physical synthesis for fitting, the Quartus II software makes placement-specific changes to the netlist that reduce resource utilization for a specific Altera device.

The compilation time might increase considerably when you use physical synthesis options.

With the Quartus II software, you can apply physical synthesis options to specific instances, which can reduce the impact on compilation time. Physical synthesis instance assignments allow you to enable physical synthesis algorithms for specific portions of their design.

The following physical synthesis optimizations for fitting are available:

- Physical synthesis for combinational logic
- Map logic into memory

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

**Retarget or Balance DSP Blocks**

A design might not fit because it requires too many DSP blocks. All DSP block functions can be implemented with logic cells, so you can retarget some of the DSP blocks to logic to obtain a fit.

If the DSP function was created with the MegaWizard Plug-In Manager, open the MegaWizard Plug-In Manager and edit the function so it targets logic cells instead of DSP blocks. The Quartus II software uses the `DEDICATED_MULTIPLIER_CIRCUITRY` megafunction parameter to control the implementation.

DSP blocks also can be inferred from your HDL code for multipliers, multiply-adders, and multiply-accumulators. This inference can be turned off in your synthesis tool. When you are using Quartus II integrated synthesis, you can disable inference by turning off the *Auto DSP Block Replacement* logic option for your entire project. On the Assignments menu, click *Settings*. In the *Category* list, select *Analysis & Synthesis Settings*, click *More Settings*, and turn off *Auto DSP Block Replacement*. Alternatively, you can disable the option for a specific block with the Assignment Editor.

For more information about disabling DSP block inference in other synthesis tools, refer to the appropriate chapter in *Section III. Synthesis* in volume 1 of the *Quartus II Handbook*, or your synthesis software’s documentation.
The Quartus II software also offers the **DSP Block Balancing** logic option, which implements DSP block elements in logic cells or in different DSP block modes. The default **Auto** setting allows DSP block balancing to convert the DSP block slices automatically as appropriate to minimize the area and maximize the speed of the design. You can use other settings for a specific node or entity, or on a project-wide basis, to control how the Quartus II software converts DSP functions into logic cells and DSP blocks. Using any value other than **Auto** or **Off** overrides the **DEDICATED_MULTIPLIER_CIRCUITRY** parameter used in megafunction variations.

For more details about the Quartus II logic options described in this section, refer to *Auto DSP Block Replacement* and *DSP Block Balancing* in Quartus II Help.

### Use a Larger Device

If a successful fit cannot be achieved because of a shortage of LEs or ALMs, memory, or DSP blocks, you might require a larger device.

### Routing

Use the suggestions in the following subsections to help you resolve routing resource problems.

**Set Auto Packed Registers to Sparse or Sparse Auto**

This option is useful for reducing LE or ALM count in a design. This option is available for all Altera devices supported by the Quartus II software.

This option can be set in the Assignment Editor, or you can set this option by clicking **More Settings** on the **Fitter Settings** page in the **Settings** dialog box.

For more information, refer to *Auto Packed Registers* in Quartus II Help.

**Set Fitter Aggressive Routability Optimizations to Always**

Use this option if your design does not fit due to excessive routing wire utilization.

For more information, refer to *Fitter Aggressive Routability Optimizations logic option* in Quartus II Help.

If there is a significant imbalance between placement and routing time (during the first fitting attempt), it might be because of high wire utilization. By turning on this option, you might be able to reduce your compilation time.

On average, this option can save up to 6% wire utilization, but can also reduce performance by up to 4%, depending on the device.

These optimizations are used automatically when the Fitter performs more than one fitting attempt, but turning the option on increases the optimization effort on the first fitting attempt. This option also ensures that the Quartus II software uses maximum optimization to reduce routability, even if the Fitter Effort is set to **Auto Fit**.

**Increase Placement Effort Multiplier**

Increasing the placement effort can improve the routability of the design, allowing the software to route a design that otherwise requires too many routing resources.
For more information refer to Placement Effort Multiplier logic option in Quartus II Help.

Increased effort is used automatically when the Fitter performs more than one fitting attempt. Setting a multiplier higher than one (before compilation) increases the optimization effort on the first fitting attempt. The second and third fitting loops increase the Placement Effort Multiplier to 4 and then to 16. These loops result in increased compilation times, with possible improvement in the quality of placement.

You can modify the Placement Effort Multiplier using the following Tcl command:
```
set_global_assignment -name PLACEMENT_EFFORT_MULTIPLIER <value>
```

<value> can be any positive, non-zero number.

Increasing placement effort is likely to reduce congestion during routing, and help fit hard-to-route designs. Increasing the Placement Effort Multiplier and limiting the Fitter to one fitting attempt for hard-to-fit designs can produce better Fitter results with lower overall compilation time.

Increase Router Effort Multiplier

The Router Effort Multiplier controls how quickly the router tries to find a valid solution. The default value is 1.0 and legal values must be greater than 0. Numbers higher than 1 help designs that are difficult to route by increasing the routing effort. Numbers closer to 0 (for example, 0.1) can reduce router runtime, but usually reduce routing quality slightly. Experimental evidence shows that a multiplier of 3.0 reduces overall wire usage by about 2%. Using a Router Effort Multiplier higher than the default value could be beneficial for designs with complex datapaths with more than five levels of logic. However, congestion in a design is primarily due to placement, and increasing the Router Effort Multiplier does not necessarily reduce congestion.

For more information, refer to Router Effort Multiplier logic option in Quartus II Help.

Remove Fitter Constraints

A design with conflicting constraints or constraints that are difficult to meet may not fit the targeted device. This can occur when location or LogicLock assignments are too strict and there are not enough routing resources.

In this case, use the Routing Congestion task in the Chip Planner to locate routing problems in the floorplan, then remove all location and LogicLock region assignments from that area. If the local constraints are removed, and the design still does not fit, the design is over-constrained. To correct the problem, remove all location and LogicLock assignments and run successive compilations, incrementally constraining the design before each compilation. You can delete specific location assignments in the Assignment Editor or the Chip Planner. Remove LogicLock assignments in the Chip Planner, in the LogicLock Regions Window, or on the Assignments menu, click Remove Assignments. Turn on the assignment categories you want to remove from the design in the Available assignment categories list.

For more information about the Routing Congestion task in the Chip Planner, refer to the Analyzing and Optimizing the Design Floorplan chapter in volume 2 of the Quartus II Handbook. You can also refer to About the Chip Planner in Quartus II Help.
**Optimize Synthesis for Area, Not Speed**

In some cases, resynthesizing the design to improve the area utilization can also improve the routability of the design. First, ensure that you have set your device and timing constraints correctly in your synthesis tool. Ensure that you do not over-constrain the timing requirements for the design, particularly when the area utilization of the design is a concern. Synthesis tools generally try to meet the specified requirements, which can result in higher device resource usage if the constraints are too aggressive.

If resource utilization is important to improving the routing results in your design, some synthesis tools offer an easy way to optimize for area instead of speed. If you are using Quartus II integrated synthesis, on the Assignments menu, click **Settings**. In the **Category** list, select **Analysis & Synthesis Settings**, and select **Balanced** or **Area** under **Optimization Technique**.

You can also specify this logic option for specific modules in your design with the Assignment Editor in cases where you want to reduce area using the **Area** setting (potentially at the expense of register-to-register timing performance). You can apply the setting to specific modules while leaving the default **Optimization Technique** setting at **Balanced** (for the best trade-off between area and speed for certain device families) or **Speed**. You can also use the **Speed Optimization Technique for Clock Domains** logic option to specify that all combinational logic in or between the specified clock domain(s) is optimized for speed.

In the Quartus II software, the **Balanced** setting typically produces utilization results that are very similar to those obtained with the **Area** setting, with better performance results. The **Area** setting can yield better results in some unusual cases.

In some synthesis tools, not specifying an f\(_{\text{MAX}}\) requirement can result in less resource utilization, which can improve routability.

For information about setting timing requirements and synthesis options in Quartus II integrated synthesis and other synthesis tools, refer to the appropriate chapter in **Section III. Synthesis** in volume 1 of the **Quartus II Handbook**, or your synthesis software’s documentation.

**Optimize Source Code**

If your design does not fit because of routing problems and the methods described in the preceding sections do not sufficiently improve the routability of the design, modify the design at the source to achieve the desired results. You can often improve results significantly by making design-specific changes to your source code, such as duplicating logic or changing the connections between blocks that require significant routing resources.

**Use a Larger Device**

If a successful fit cannot be achieved because of a shortage of routing resources, you might require a larger device.
Timing Optimization Techniques (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

Beginning with the Quartus II software version 10.1, a new 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 Arria II GX, Arria II GZ, Cyclone III, Cyclone IV, Stratix III, Stratix IV, and Stratix V device families.

Selecting the Report Timing Closure Recommendations task opens the Report Design Analysis dialog box (Figure 13–5).

Figure 13–5. Report Design Analysis Dialog Box

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 dialog box (Figure 13–5), you can select paths based on the clock domain, filter by nodes on path, and choose the number of paths to analyze.
After running this command 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.

Figure 13–6 shows an example report.

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 a 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 appropriate action if user constraints cause a specific placement.

**Timing Optimization Advisor**

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 on 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.

The example in Figure 13–7 shows the Timing Optimization Advisor after compiling a design that meets its frequency requirements, but requires setting changes to improve the timing.

**Figure 13–7. Timing Optimization Advisor**

When you expand one of the categories in the 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 13–7, 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 Advisor.

There is a link from each recommendation to the appropriate location in the Quartus II UI 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 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**

The next stage of design optimization focuses on I/O timing. Ensure that you have made the appropriate assignments as described in “Initial Compilation: Required Settings” on page 13–2, and that the 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 the “Register-to-Register Timing Optimization Techniques (LUT-Based Devices)” on page 13–33.

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 13–1 shows 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>Technique</th>
<th>Affects $t_{SU}$</th>
<th>Affects $t_{CO}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ensure that the appropriate constraints are set for the failing I/Os (page 13–3)</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Use timing-driven compilation for I/O (page 13–30)</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Use fast input register (page 13–31)</td>
<td>✓</td>
<td>—</td>
</tr>
<tr>
<td>Use fast output register, fast output enable register, and fast OCT register (page 13–31)</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>
<tr>
<td>Decrease the value of Input Delay from Pin to Internal Cells, or set Decrease Input Delay to Internal Cells = 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

You can place individual registers in I/O cells manually by making fast I/O assignments with the Assignment Editor. For an input register, use the **Fast Input Register** option; for an output register, use the **Fast Output Register** option; and for an output enable register, use the **Fast Output Enable Register** option. Stratix II devices also support the **Fast OCT (on-chip termination) Register** option. 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.

### Table 13-1. Improving Setup and Clock-to-Output Times (*Note 1*) (Part 2 of 2)

<table>
<thead>
<tr>
<th>Technique</th>
<th>Affects ( t_{SU} )</th>
<th>Affects ( t_{CO} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decrease the value of <em>Delay from Output Register to Output Pin</em>, or set <em>Increase Delay to Output Pin</em> = OFF (page 13–32)</td>
<td>—</td>
<td>✓</td>
</tr>
<tr>
<td>Increase the value of <em>Input Delay from Dual-Purpose Clock Pin to Fan-Out Destinations</em> (page 13–32)</td>
<td>✓</td>
<td>—</td>
</tr>
<tr>
<td>Use PLLs to shift clock edges (page 13–32)</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Use the Fast Regional Clock (page 13–33)</td>
<td>—</td>
<td>✓</td>
</tr>
<tr>
<td>For MAX II series devices, set <strong>Guarantee I/O paths to zero</strong>, <strong>Hold Time at Fast Timing Corner to OFF</strong>, or when ( t_{SU} ) and ( t_{CO} ) constraints permit (page 13–33)</td>
<td>✓</td>
<td>—</td>
</tr>
<tr>
<td>Increase the value of <em>Delay to output enable pin</em> or set <em>Increase delay to output enable pin</em> (page 13–32)</td>
<td>—</td>
<td>✓</td>
</tr>
</tbody>
</table>

**Note to Table 13–1:**

1. These options may not apply to all device families.
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 details 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 in volume 2 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
- Input Delay from Pin to Input Register
- Decrease Input Delay to Internal Cells
- Input Delay from Pin to Internal Cells
- Decrease Input Delay to Output Register
- Increase Delay to Output Enable Pin
- Output Enable Pin Delay
- Increase Delay to Output Pin
- Delay from Output Register to Output Pin
- Increase Input Clock Enable Delay
- Input Delay from Dual-Purpose Clock Pin to Fan-Out Destinations
- Increase Output Clock Enable Delay
- Increase Output Enable Clock Enable Delay
- Increase $t_{zx}$ Delay to Output Pin

**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 $t_{PH}$ at the expense of $t_{SU}$, while shifting it forward gives a better $t_{SL}$ at the expense of $t_{PH}$ (refer to Figure 13–8). This technique can be used only in devices that offer PLLs with the phase shift option.

**Figure 13–8. Shift Clock Edges Forward to Improve $t_{SU}$ at the Expense of $t_{PH}$**

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** 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 (GCLKs), regional clock networks (RCLKs), fast regional clock networks (FCLK) and periphery clock networks (PCLKs). The available resources differ between various Altera device families.

For the number of various 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.
Change How Hold Times are Optimized for MAX II Devices

For MAX II series devices, you can use the **Guarantee I/O paths have zero hold time at Fast Timing 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 in volume 1 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 for the State Machine report under Analysis & Synthesis in the Compilation Report. This report provides details, including the state encoding for each state machine that was recognized during compilation. If your state machine is not being 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 in volume 1 of the Quartus II Handbook.

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

### 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 “Timing Requirement Settings” on page 13–3.

2. Ensure that you have reviewed all warning messages from your initial compilation and check for ignored timing assignments. Refer to “Design Analysis” on page 13–9 for details and fix any of these problems before proceeding with optimization.

3. Apply netlist synthesis optimization options.

   Apply the following synthesis options to optimize for speed:
   - “Optimize Synthesis for Speed, Not Area” on page 13–36
   - “Flatten the Hierarchy During Synthesis” on page 13–37
   - “Set the Synthesis Effort to High” on page 13–37
   - “Change State Machine Encoding” on page 13–38
   - “Prevent Shift Register Inference” on page 13–38
   - “Use Other Synthesis Options Available in Your Synthesis Tool” on page 13–39

4. Apply the following options for physical synthesis optimization:

   - Perform physical synthesis for combinational logic
   - Perform automatic asynchronous signal pipelining
   - Perform register duplication
   - Perform register retiming
   - Perform logic to memory mapping

5. Try different Fitter seeds (page 13–39). You can omit this step if a large number of critical paths are failing, or if the paths are failing badly.
6. Make LogicLock assignments (page 13–40) to control placement.

7. Make design source code modifications to fix areas of the design that are still failing timing requirements by significant amounts (page 13–33).

8. Make location assignments, or as a last resort, perform manual placement by back-annotating the design (page 13–41).

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 13–33).

**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 for **Speed**, **Area**, or **Balanced**, 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

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

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

Follow these steps:
1. In the Assignment Editor, indicate the module instance you want to apply to the specific physical synthesis setting in the To tab.
2. Select the required physical synthesis assignment in the Assignment Name tab.
3. In the Value tab, select ON.
4. In the Enabled tab, select Yes.

**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 in volume 2 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 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 information about setting timing requirements and synthesis options in Quartus II integrated synthesis and third-party synthesis tools, refer to the appropriate chapter in Section III, Synthesis in volume 1 of the Quartus II Handbook, or refer to your synthesis software documentation.

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 in volume 1 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.

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:

```
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 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, using LogicLock assignments to improve timing performance is not recommended. For these devices, the LogicLock feature is intended to be used 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 to see 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 over-constraint. 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 chapter in volume 2 of the Quartus II Handbook.

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 13–9, which has failing paths in the timing analysis results similar to those shown in Table 13–2, 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 13–9. Design Hierarchy

Table 13–2 shows the failing paths connecting two regions together within mod_A listed in the timing analysis report.

Table 13–2. 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 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 in volume 1 of the Quartus II Handbook, and Analyzing and Optimizing the Design Floorplan chapter in volume 2 of the Quartus II Handbook.

Location Assignments and Back-Annotation

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 failure (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 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 in volume 1 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.

**Resource Utilization Optimization Techniques (Macrocell-Based CPLDs)**

The following recommendations help you take advantage of the macrocell-based architecture in the MAX 7000 and MAX 3000 devices to yield maximum speed, reliability, and device resource utilization while minimizing fitting difficulties.

After design analysis, the first stage of design optimization is to improve resource utilization. Complete this stage before proceeding to timing optimization. First, ensure that you have set the basic constraints described in “Initial Compilation: Required Settings” on page 13–2. If your design is not fitting into a specified device, use the techniques in this section to achieve a successful fit.

**Use Dedicated Inputs for Global Control Signals**

MAX 7000 and MAX 3000 devices have four dedicated inputs that can be used for global register control. Because the global register control signals can bypass the logic cell array and directly feed registers, product terms can be preserved for primary logic. Also, because each signal has a dedicated path into the LAB, global signals also can bypass logic and data path interconnect resources.
Because the dedicated input pins are designed for high fan-out control signals and provide low skew, always assign global signals (such as clock, clear, and output enable) to the dedicated input pins.

You can use logic-generated control signals for global control signals instead of dedicated inputs. However, the following list shows the disadvantages of using logic-generated control signals:

- More resources are required (logic cells, interconnect).
- More data skew is introduced.
- If the logic-generated control signals have high fan-out, the design can be more difficult to fit.

By default, the Quartus II software uses dedicated inputs for global control signals automatically. You can assign control signals to dedicated input pins in one of the following ways:

- In the Assignment Editor, select one of the two following methods:
  - Assign pins to dedicated pin locations.
  - Assign a Global Signal setting to the pins.

- On the Assignments menu, click Settings. On the Analysis & Synthesis Settings page, click More Settings, and in the Existing Option settings section, select Auto Global Register Control Signals.

- Insert a GLOBAL primitive after the pins.

- If you have already assigned pins for the design in the MAX+PLUS® II software, on the Assignments menu, click Import Assignments.

**Reserve Device Resources**

Because pin and logic option assignments can be necessary for board layout and performance requirements, and because full utilization of the device resources can increase the difficulty of fitting the design, Altera recommends that you leave 10% of the logic cells and 5% of the I/O pins unused to accommodate future design modifications. Following the Altera-recommended device resource reservation guidelines for macrocell-based CPLDs increases the chance that the Quartus II software can fit the design during recompilation after changes or assignments have been made.

**Pin Assignment Guidelines and Procedures**

Sometimes user-specified pin assignments are necessary for board layout. This section discusses pin assignment guidelines and procedures.

To minimize fitting issues with pin assignments, follow these guidelines:

- Assign speed-critical control signals to dedicated inputs.
- Assign output enables to appropriate locations.
- Estimate fan-in to assign output pins to the appropriate LAB.
- Assign output pins that require parallel expanders to macrocells numbered 4 to 16.
Altera recommends that you allow the Quartus II software to select pin assignments automatically when possible. You can use the Quartus II Pin Advisor feature (accessible from the Tools menu) for pin connection guidelines.

For more information about the Pin Advisor, refer to Pin Advisor Command in Quartus II Help.

**Control Signal Pin Assignments**

Assign speed-critical control signals to dedicated input pins. Every MAX 7000 and MAX 3000 device has four dedicated input pins (GCLK1, OE2/GCLK2, OE1, and GCLRn). You can assign clocks to global clock dedicated inputs (GCLK1 and OE2/GCLK2), clear to the global clear dedicated input (GCLRn), and speed-critical output enable to global OE dedicated inputs (OE1 and OE2/GCLK2).

**Output Enable Pin Assignments**

Occasionally, because the total number of required output enable pins is more than the dedicated input pins, output enable signals must be assigned to I/O pins.

To minimize possible fitting errors when assigning the output enable pins for MAX 7000 and MAX 3000 devices, refer to Pin-Out Files for Altera Devices on the Altera website (www.altera.com).

**Estimate Fan-In When Assigning Output Pins**

Macrocells with high fan-in can cause more placement problems for the Quartus II Fitter than those with low fan-in. The maximum fan-in per LAB should not exceed 36 in MAX 7000 and MAX 3000 devices. Therefore, estimate the fan-in of logic (such as an x-input AND gate) that feeds each output pin. If the total fan-in of logic that feeds each output pin in the same LAB exceeds 36, compilation can fail. To save resources and prevent compilation errors, avoid assigning pins that have high fan-in.

**Outputs Using Parallel Expander Pin Assignments**

Figure 13–10 illustrates how parallel expanders are used within a LAB. MAX 7000 and MAX 3000 devices contain chains that can lend or borrow parallel expanders. The Quartus II Fitter places macrocells in a location that allows them to lend and borrow parallel expanders appropriately.
As shown in Figure 13–10, only macrocells 2 through 16 can borrow parallel expanders. Therefore, assign output pins that might require parallel expanders to pins adjacent to macrocells 4 through 16. Altera recommends using macrocells 4 through 16 because they can borrow the largest number of parallel expanders.

![Figure 13–10. LAB Macrocells and Parallel Expander Associations](image)

**Resolving Resource Utilization Problems**

Two common Quartus II compilation fitting issues cause errors: excessive macrocell usage and lack of routing resources. Macrocell usage errors occur when the total number of macrocells in the design exceed the available macrocells in the device. Routing errors occur when the available routing resources are insufficient to implement the design. Check the Message window for the compilation results.

Messages in the Messages window are also copied in the Report Files. Right-click on a message and click **Help** for more information.
Resolving Macrocell Usage Issues

Occasionally, a design requires more macrocell resources than are available in the selected device, which results in the design not fitting. The following list provides tips for resolving macrocell usage issues as well as tips to minimize the number of macrocells used:

- On the Assignments menu, click Settings. In the Category list, select Analysis & Synthesis Settings, click More Settings, and turn off Auto Parallel Expanders. If the design’s clock frequency ($f_{\text{MAX}}$) is not an important design requirement, turn off parallel expanders for all or part of the project. The design usually requires more macrocells if parallel expanders are turned on.

- Change Optimization Technique from Speed to Area. Selecting Area instructs the compiler to give preference to area utilization rather than speed ($f_{\text{MAX}}$). On the Assignments menu, click Settings. In the Category list, change the Optimization Technique option in the Analysis & Synthesis Settings page.

- Use D-type flipflops instead of latches. Altera recommends that you always use D-type flipflops instead of latches in your design because D-type flipflops can reduce the macrocell fan-in, and thus reduce macrocell usage. The Quartus II software uses extra logic to implement latches in MAX 7000 and MAX 3000 designs because MAX 7000 and MAX 3000 macrocells contain D-type flipflops instead of latches.

- Use asynchronous clear and preset instead of synchronous clear and preset. To reduce the product term usage, use asynchronous clear and preset in your design whenever possible. Using other control signals such as synchronous clear produces macrocells and pins with higher fan-out.

After following the suggestions in this section, if your project still does not fit the targeted device, consider using a larger device. When upgrading to a different density, the vertical package-migration feature of the MAX 7000 and MAX 3000 device families allows pin assignments to be maintained.

Resolving Routing Issues

Routing is another resource that can cause design fitting issues. For example, if the total fan-in into a LAB exceeds the maximum allowed, a no-fit error can occur during compilation. If your design does not fit the targeted device because of routing issues, consider the following suggestions:

- Use dedicated inputs/global signals for high fan-out signals. The dedicated inputs in MAX 7000 and MAX 3000 devices are designed for speed-critical and high fan-out signals. Always assign high fan-out signals to dedicated inputs/global signals.

- Change the Optimization Technique option from Speed to Area. This option can resolve routing resource and macrocell usage issues. Refer to “Resolving Macrocell Usage Issues” on page 13–46.
Reduce the fan-in per cell. If you are not limited by the number of macrocells used in the design, you can use the Fan-in per cell (%) option to reduce the fan-in per cell. The allowable values are 20–100%; the default value is 100%. Reducing the fan-in can reduce localized routing congestion but increase the macrocell count. You can set this logic option in the Assignment Editor or under More Settings in the Analysis & Synthesis Settings page of the Settings dialog box.

On the Assignments menu, click Settings. In the Category list, select Analysis & Synthesis Settings, click More Options, and turn off Auto Parallel Expanders. By turning off the parallel expanders, you give the Quartus II software more fitting flexibility for each macrocell, allowing macrocells to be relocated. For example, each macrocell (previously grouped together in the same LAB) can be moved to a different LAB to reduce routing constraints.

Insert logic cells. Inserting logic cells reduces fan-in and shared expanders used per macrocell, increasing routability. By default, the Quartus II software automatically inserts logic cells when necessary. Otherwise, Auto Logic Cell can be disabled as follows. On the Assignments menu, click Settings. In the Category list, select Analysis & Synthesis Settings. Under More Settings, turn off Auto Logic Cell Insertion. Refer to “Using LCELL Buffers to Reduce Required Resources” for more information.

Change pin assignments. If you want to discard your pin assignments, you can let the Quartus II Fitter ignore some or all of the assignments.

If you prefer reassigning pins to increase routing efficiency, refer to “Pin Assignment Guidelines and Procedures” on page 13–43.

Using LCELL Buffers to Reduce Required Resources

Complex logic, such as multilevel XOR gates, are often implemented with more than one macrocell. When this occurs, the Quartus II software automatically allocates shareable expanders—or additional macrocells (called synthesized logic cells)—to supplement the logic resources that are available in a single macrocell. You can also break down complex logic by inserting logic cells in the project to reduce the average fan-in and the total number of shareable expanders required. Manually inserting logic cells can provide greater control over speed-critical paths.

Instead of using the Quartus II software’s Auto Logic Cell Insertion option, you can manually insert logic cells. However, Altera recommends that you use the Auto Logic Cell Insertion option unless you know which part of the design is causing the congestion.

A good location to manually insert LCELL buffers is where a single complex logic expression feeds multiple destinations in your design. You can insert an LCELL buffer just after the complex expression; the Quartus II Fitter extracts this complex expression and places it in a separate logic cell. Rather than duplicate all the logic for each destination, the Quartus II software feeds the single output from the logic cell to all destinations.
To reduce fan-in and prevent no-fit compilations caused by routing resource issues, insert an LCELL buffer after a NOR gate (Figure 13–11). The design in Figure 13–11 was compiled for a MAX 7000AE device. Without the LCELL buffer, the design requires two macrocells and eight shareable expanders, and the average fan-in is 14.5 macrocells. However, with the LCELL buffer, the design requires three macrocells and eight shareable expanders, and the average fan-in is just 6.33 macrocells.

Figure 13–11. Reducing the Average Fan-In by Inserting LCELL Buffers

Timing Optimization Techniques (Macrocell-Based CPLDs)

After resource optimization, design optimization focuses on timing. Ensure that you have made the appropriate assignments as described in “Initial Compilation: Required Settings” on page 13–2, and that the resource utilization is satisfactory before proceeding with timing optimization.

The following five timing parameters are primarily responsible for a design’s performance:

- Setup time \( (t_{SU}) \) —the propagation time for input data signals
- Hold time \( (t_{H}) \) —the propagation time for input data signals
- Clock-to-output time \( (t_{CO}) \) —the propagation time for output signals
- Pin-to-pin delays \( (t_{PD}) \) —the time required for a signal from an input pin to propagate through combinational logic and appear at an external output pin
- Maximum clock frequency \( (f_{MAX}) \) —the internal register-to-register performance
This section provides guidelines to improve the timing if the timing requirements are not met. Figure 13–12 shows the parts of the design that determine the $t_{SU}$, $t_H$, $t_{CO}$, $t_{PD}$, and $f_{MAX}$ timing parameters.

**Figure 13–12. Main Timing Parameters that Determine the System’s Performance**

When you are analyzing a design to improve performance, be sure to consider the two major contributors to long delay paths:

- Excessive levels of logic
- Excessive loading (high fan-out)

When a MAX 7000 or MAX 3000 device signal drives more than one LAB, the programmable interconnect array (PIA) delay increases by 0.1 ns per additional LAB fan-out. Therefore, to minimize the added delay, concentrate the destination macrocells into fewer LABs, minimizing the number of LABs that are driven. The main cause of long delays in circuit design is excessive levels of logic.

### Improving Setup Time

Sometimes the $t_{SU}$ timing reported by the Quartus II Fitter does not meet your timing requirements. To improve the $t_{SU}$ timing, refer to the following guidelines:

- Turn on the **Fast Input Register** option using the Assignment Editor. The **Fast Input Register** option allows input pins to directly drive macrocell registers via the fast-input path, thus minimizing the pin-to-register delay. This option is useful when a pin drives a D-type flipflop and there is no combinational logic between the pin and the register.

- Reduce the amount of logic between the input and the register. Excessive logic between the input pin and register causes more delays. To improve setup time, Altera recommends that you reduce the amount of logic between the input pin and the register whenever possible.

- Reduce fan-out. The delay from input pins to macrocell registers increases when the fan-out of the pins increases. To improve the setup time, minimize the fan-out.
### Improving Clock-to-Output Time

To improve a design's clock-to-output time, minimize the register-to-output-pin delay. To improve the $t_{CO}$ timing, refer to the following guidelines:

- **Use the global clock.** In addition to minimizing the delay from a register to an output pin, minimizing the delay from the clock pin to the register can also improve $t_{CO}$ timing. Always use the global clock for low-skew and speed-critical signals.

- **Reduce the amount of logic between the register and output pin.** Excessive logic between the register and the output pin causes more delay. Always minimize the amount of logic between the register and output pin for faster clock-to-output time.

Table 13–3 shows the timing results for an EPM7064AETC100-4 device when a combination of the Fast Input Register option, global clock, and minimal logic is used. When the Fast Input Register option is turned on, the $t_{SU}$ timing is improved ($t_{SU}$ decreases from 1.6 ns to 1.3 ns and from 2.8 ns to 2.5 ns). The $t_{CO}$ timing is improved when the global clock is used for low-skew and speed-critical signals ($t_{CO}$ decreases from 4.3 ns to 3.1 ns). However, if there is additional logic used between the input pin and the register or the register and the output pin, the $t_{SU}$ and $t_{CO}$ delays increase.

### Table 13–3. EPM7064AETC100-4 Device Timing Results

<table>
<thead>
<tr>
<th>Number of Registers</th>
<th>$t_{SU}$ (ns)</th>
<th>$t_{H}$ (ns)</th>
<th>$t_{CO}$ (ns)</th>
<th>Global Clock Used</th>
<th>Fast Input Register Option</th>
<th>D Input Location</th>
<th>Q Output Location</th>
<th>Additional Logic Between:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>D Input Location &amp; Register</td>
</tr>
<tr>
<td>1</td>
<td>1.3</td>
<td>1.2</td>
<td>4.3</td>
<td>—</td>
<td>On</td>
<td>LAB A</td>
<td>LAB A</td>
<td>—</td>
</tr>
<tr>
<td>1</td>
<td>1.6</td>
<td>0.3</td>
<td>4.3</td>
<td>—</td>
<td>Off</td>
<td>LAB A</td>
<td>LAB A</td>
<td>—</td>
</tr>
<tr>
<td>1</td>
<td>2.5</td>
<td>0</td>
<td>3.1</td>
<td>✓</td>
<td>On</td>
<td>LAB A</td>
<td>LAB A</td>
<td>—</td>
</tr>
<tr>
<td>1</td>
<td>2.8</td>
<td>0</td>
<td>3.1</td>
<td>✓</td>
<td>Off</td>
<td>LAB A</td>
<td>LAB A</td>
<td>—</td>
</tr>
<tr>
<td>1</td>
<td>3.6</td>
<td>0</td>
<td>3.1</td>
<td>✓</td>
<td>Off</td>
<td>LAB A</td>
<td>LAB A</td>
<td>✓</td>
</tr>
<tr>
<td>1</td>
<td>2.8</td>
<td>0</td>
<td>7.0</td>
<td>✓</td>
<td>Off</td>
<td>LAB D</td>
<td>LAB A</td>
<td>—</td>
</tr>
<tr>
<td>16 with the same D and clock inputs</td>
<td>2.8</td>
<td>0</td>
<td>All 6.2</td>
<td>✓</td>
<td>Off</td>
<td>LAB D</td>
<td>LAB A, B</td>
<td>—</td>
</tr>
<tr>
<td>32 with the same D and clock inputs</td>
<td>2.8</td>
<td>0</td>
<td>All 6.4</td>
<td>✓</td>
<td>Off</td>
<td>LAB C</td>
<td>LAB A, B, C</td>
<td>—</td>
</tr>
</tbody>
</table>
Improving Propagation Delay (t\text{PD})

Achieving fast propagation delay (t\text{PD}) timing is required in many system designs. However, if there are long delay paths through complex logic, achieving fast propagation delays can be difficult. To improve your design’s t\text{PD}, refer to the following guidelines:

- On the Assignments menu, click Settings. In the Category list, select Analysis & Synthesis Settings, and turn on Auto Parallel Expanders. Turning on the parallel expanders for individual nodes or sub-designs can increase the performance of complex logic functions. However, if the project’s pin or logic cell assignments use parallel expanders placed physically together with macrocells (which can reduce routability), parallel expanders can cause the Quartus II Fitter to have difficulties finding and optimizing a fit. Additionally, the number of macrocells required to implement the design increases and results in a no-fit error during compilation if the device resources are limited. For more information about turning on the Auto Parallel Expanders option, refer to “Resolving Macrocell Usage Issues” on page 13–46.

- Set the Optimization Technique to Speed. By default, the Quartus II software sets the Optimization Technique option to Speed for MAX 7000 and MAX 3000 devices. Reset the Optimization Technique option to Speed only if you previously set it to Area. On the Assignments menu, click Settings. In the Category list, select Analysis & Synthesis Settings, and turn on Speed under Optimization Technique.

Improving Maximum Frequency (f\text{MAX})

Maintaining the system clock at or above a certain frequency is a major goal in circuit design. For example, if you have a fully synchronous system that must run at 100 MHz, the longest delay path from the output of any register to the inputs of the registers it feeds must be less than 10 ns. Maintaining the system clock speed can be difficult if there are long delay paths through complex logic. Altera recommends that you perform the following guidelines to increase your design’s clock speed (f\text{MAX}):

- On the Assignments menu, click Settings. In the Category list, select Analysis & Synthesis Settings, click More Settings, and turn on Auto Parallel Expanders. Turning on the parallel expanders for individual nodes or subdesigns can increase the performance of complex logic functions. However, if the project’s pin or logic cell assignments use parallel expanders placed physically together with macrocells (which can reduce routability), parallel expanders can cause the Quartus II compiler to have difficulties finding and optimizing a fit. Additionally, the number of macrocells required to implement the design also increases and can result in a no-fit error during compilation if the device’s resources are limited. For more information about using the Auto Parallel Expanders option, refer to “Resolving Macrocell Usage Issues” on page 13–46.

- Use global signals or dedicated inputs. Altera MAX 7000 and MAX 3000 devices have dedicated inputs that provide low skew and high speed for high fan-out signals. Minimize the number of control signals in the design and use the dedicated inputs to implement them.
Set the Optimization Technique to Speed. By default, the Quartus II software sets the Optimization Technique option to Speed for MAX 7000 and MAX 3000 devices. Reset the Optimization Technique option to Speed only if you have previously set it to Area. You can reset the Optimization Technique option. In the Category list, select Analysis & Synthesis Settings, and turn on Speed under Optimization Technique.

Pipeline the design. Pipelining, which increases clock frequency (f_{\text{MAX}}), refers to dividing large blocks of combinational logic by inserting registers. When using RAM or DSP blocks, always enable the optional input and output registers.

Optimizing Source Code—Pipelining for Complex Register Logic

If the methods described in the preceding sections do not sufficiently improve your results, modify the design at the source to achieve the desired results. Using additional register stages (pipeline registers) consumes more device resources, but it also lowers the propagation delay between registers, allowing you to maintain high system clock speed.

Refer to the application note AN 584: Timing Closure Methodology for Advanced FPGA Designs for more information about pipelining registers and other examples of optimizing source code.

Other Optimization Resources

The Quartus II software has additional resources to help you optimize your design for resource, performance, compilation time, and power.

Design Space Explorer

The DSE automates the process of running multiple compilations with different settings. You can use the DSE to try the techniques described in this chapter. The DSE utility helps automate the process of finding the best set of options for your design. The DSE explores the design space by applying various optimization techniques and analyzing the results.

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

Other Optimization Advisors

The Power Optimization Advisor provides guidance for reducing power consumption. In addition, the Incremental Compilation Advisor provides suggestions to improve your results when partitioning your design for a hierarchical or team-based design flow using the Quartus II incremental compilation feature.

For more information about using the Power Optimization Advisor, refer to the Power Optimization chapter in volume 2 of the Quartus II Handbook. For more information about using the Incremental Compilation Advisor, refer to the Quartus II Incremental Compilation for Hierarchical and Team-Based Design chapter in volume 1 of the Quartus II Handbook.
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 in volume 2 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 in volume 2 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 <.qsf variable name> <value>
```

Use the following Tcl command to make an instance assignment:

```
set_instance_assignment -name <.qsf 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

The Quartus II Settings File (.qsf) variable name is used 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 13–4 lists the .qsf variable name and applicable values for the settings discussed in “Initial Compilation: Required Settings” on page 13–2. Table 13–5 shows the list of 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>Device Setting</td>
<td>DEVICE</td>
<td>&lt;device part number&gt;</td>
<td>Global</td>
</tr>
<tr>
<td>Use Smart Compilation</td>
<td>SPEED_DISK_USAGE_TRADEOFF</td>
<td>SMART, NORMAL</td>
<td>Global</td>
</tr>
<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>
<tr>
<td>Fitter Effort</td>
<td>FITTER_EFFORT</td>
<td>STANDARD FIT, FAST FIT, AUTO FIT</td>
<td>Global</td>
</tr>
</tbody>
</table>
### Table 13–5. 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 Effort Multiplier</td>
<td>ROUTER_EFFORT_MULTIPLIER</td>
<td>Any positive, non-zero value</td>
<td>Global</td>
</tr>
<tr>
<td>Router Timing Optimization level</td>
<td>ROUTER_TIMING_OPTIMIZATION_LEVEL</td>
<td>NORMAL, MINIMUM, MAXIMUM</td>
<td>Global</td>
</tr>
<tr>
<td>Final Placement Optimization</td>
<td>FINAL_PLACEMENT_OPTIMIZATION</td>
<td>ALWAYS, AUTOMATICALLY, NEVER</td>
<td>Global</td>
</tr>
</tbody>
</table>

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

Table 13–6 lists the .qsf file variable name and applicable values for the settings discussed in “Resource Utilization Optimization Techniques (LUT-Based Devices)” on page 13–15.

### Table 13–6. 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 (1)</td>
<td>AUTO_PACKED_REGISTERS_&lt;device family name&gt;</td>
<td>OFF, NORMAL, MINIMUM, AREA</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>
<tr>
<td>Speed Optimization Technique for Clock Domains</td>
<td>SYNT_CRI_TICAL_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, MINIMAL, BITS, USER-ENCODE</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_REGISTRER_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>
</tbody>
</table>
Table 13–6. 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>Number of Processors for Parallel Compilation</td>
<td>NUM_PARALLEL_PROCESSORS</td>
<td>Integer between 1 and 4 inclusive, or ALL</td>
<td>Global</td>
</tr>
</tbody>
</table>

**Note to Table 13–6:**
(1) Allowed values for this setting depend on the device family that is selected.

---

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

Table 13–7 lists the .qsf file variable name and applicable values for the I/O timing optimization settings.

Table 13–7. 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 13–8 lists the .qsf file variable name and applicable values for the settings discussed in “Register-to-Register Timing Optimization Techniques (LUT-Based Devices)” on page 13–33.

Table 13–8. Register-to-Register Timing 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>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_PIPELINE</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>
</tbody>
</table>
Conclusion

Using the recommended techniques described in this chapter can help you close timing quickly on complex designs, reduce iterations by providing more intelligent and better links between analysis and assignment tools, and balance multiple design constraints including multiple clocks, routing resources, and area constraints.

The Quartus II software provides many features to achieve optimal results. Follow the techniques presented in this chapter to efficiently optimize a design for area or timing performance, or to reduce compilation time.

Document Revision History

Table 13–9 shows the revision history for this chapter.

Table 13–8. Register-to-Register Timing 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>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>

Table 13–9. Document Revision History (Part 1 of 3)

<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                                 |
Table 13–9. Document Revision History (Part 2 of 3)

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>November 2009</td>
<td>9.1.0</td>
<td>▪ Removed unsupported Timing Closure Floorplan references</td>
</tr>
<tr>
<td></td>
<td></td>
<td>▪ Removed references to unsupported device families</td>
</tr>
<tr>
<td></td>
<td></td>
<td>▪ Added several notes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>▪ Minor text edits</td>
</tr>
<tr>
<td>March 2009</td>
<td>9.0.0</td>
<td>▪ Was chapter 8 in the 8.1.0 release.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>▪ Updated the following sections:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>▪ “Timing Analysis with the TimeQuest Timing Analyzer” on page 10–14</td>
</tr>
<tr>
<td></td>
<td></td>
<td>▪ “Perform WYSIWYG Resynthesis with Balanced or Area Setting” on page 10–22</td>
</tr>
<tr>
<td></td>
<td></td>
<td>▪ “Use Physical Synthesis Options to Reduce Area” on page 10–26</td>
</tr>
<tr>
<td></td>
<td></td>
<td>▪ “Metastability Analysis and Optimization Techniques” on page 10–32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>▪ “Use Fast Regional Clock Networks and Regional Clocks Networks” on page 10–39</td>
</tr>
<tr>
<td></td>
<td></td>
<td>▪ “Register-to-Register Timing Optimization Techniques (LUT-Based Devices)” on page 10–40</td>
</tr>
<tr>
<td></td>
<td></td>
<td>▪ “Physical Synthesis Optimizations” on page 10–41</td>
</tr>
<tr>
<td></td>
<td></td>
<td>▪ “Duplicate Logic for Fan-Out Control” on page 10–45</td>
</tr>
<tr>
<td></td>
<td></td>
<td>▪ “LogicLock Assignments” on page 10–49</td>
</tr>
<tr>
<td></td>
<td></td>
<td>▪ “Enable Beneficial Skew Optimization” on page 10–48</td>
</tr>
<tr>
<td></td>
<td></td>
<td>▪ “Use Multiple Processors for Parallel Compilation” on page 10–65</td>
</tr>
<tr>
<td></td>
<td></td>
<td>▪ Removed “Analyze Your Design for Megastability”</td>
</tr>
<tr>
<td></td>
<td></td>
<td>▪ Updated Table 10–11 and Table 10–9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>▪ Removed Tables 8-1, 8-2, 8-3, 8-6, and 8-7 from version 8.1</td>
</tr>
</tbody>
</table>
### Table 13–9. Document Revision History (Part 3 of 3)

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>November 2008</td>
<td>8.1.0</td>
<td>- Changed document to 8½” × 11” page size.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Updated the following sections:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ “Optimizing Your Design” on page 10–2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ “Timing Requirement Settings” on page 10–4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ “Optimize Hold Timing” on page 10–8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ “Limit to One Fitting Attempt” on page 10–9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ “Auto Fit” on page 10–10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ “Fast Fit” on page 10–11</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ “Ignored Timing Assignments” on page 10–12</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ “I/O Timing (Including I/PD)” on page 10–13</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ “Register-to-Register Timing” on page 10–14</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ “Timing Analysis with the TimeQuest Timing Analyzer” on page 10–14</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ “Use I/O Assignment Analysis” on page 10–20</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ “Flatten the Hierarchy During Synthesis” on page 10–25</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ “Retarget Memory Blocks” on page 10–25</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ “Use Physical Synthesis Options to Reduce Area” on page 10–26</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ “Increase Placement Effort Multiplier” on page 10–30</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ “Metastability Analysis and Optimization Techniques” on page 10–32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ “Synthesis Netlist Optimizations and Physical Synthesis Optimizations” on page 10–43</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ “Incremental Compilation” on page 10–65</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ “Use Multiple Processors for Parallel Compilation” on page 10–66</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Updated Table 10–9 on page 10–73 and Table 10–11 on page 10–75</td>
</tr>
<tr>
<td>May 2008</td>
<td>8.0.0</td>
<td>- Updated links</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Updated the following sections:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Other Optimization Resources]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Setting Process Priority</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Location Assignment and Back-Annotation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Fitter Effort Setting</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Synthesis Netlist Optimizations and Physical Synthesis Optimizations</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Fast Fit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Added Metastability Analysis</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Added Enable Beneficial Skew Optimization and Analyze Your Design for Metastability</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Removed figures from “Optimizing Source Code—Pipelining for Complex Register Logic”</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Updated Table 8-5</td>
</tr>
</tbody>
</table>

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

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

The Quartus® II software offers power-driven compilation to fully optimize device power consumption. Power-driven compilation focuses on reducing your design’s total power consumption using power-driven synthesis and power-driven place-and-route. This chapter describes the power-driven compilation feature and flow in detail, as well as low power design techniques that can further reduce power consumption in your design. The techniques primarily target Arria® GX, Stratix® and Cyclone® series of devices, and HardCopy® II devices. These devices utilize a low-k dielectric material that dramatically reduces dynamic power and improves performance. Arria series, Stratix II, Stratix III, Stratix IV, and Stratix V device families include efficient logic structures called adaptive logic modules (ALMs) that obtain maximum performance while minimizing power consumption. Cyclone device families offer the optimal blend of high performance and low power in a low-cost FPGA.

For more information about a device-specific architecture, refer to the device handbook, available from the Literature and Technical Documentation page on the Altera website.

Altera provides the Quartus II PowerPlay Power Analyzer to aid you during the design process by delivering fast and accurate estimations of power consumption. You can minimize power consumption, while taking advantage of the industry’s leading FPGA performance, by using the tools and techniques described in this chapter.

For more information about the PowerPlay Power Analyzer, refer to the PowerPlay Power Analysis chapter in volume 3 of the Quartus II Handbook.

Total FPGA power consumption is comprised of I/O power, core static power, and core dynamic power. This chapter focuses on design optimization options and techniques that help reduce core dynamic power and I/O power. In addition to these techniques, there are additional power optimization techniques available for Stratix III and Stratix IV devices. These techniques include:

- Selectable Core Voltage (available only for Stratix III devices)
- Programmable Power Technology
- Device Speed Grade Selection

For more information about power optimization techniques available for Stratix III devices, refer to AN 437: Power Optimization in Stratix III FPGAs. For more information about power optimization techniques available for Stratix IV devices, refer to AN 514: Power Optimization in Stratix IV FPGAs.
Power Dissipation

This section describes the sources of power dissipation in Stratix III and Cyclone III devices. You can refine techniques that reduce power consumption in your design by understanding the sources of power dissipation.

Figure 14–1 shows the power dissipation of Stratix III and Cyclone III devices in different designs. All designs were analyzed at a fixed clock rate of 100 MHz and exhibited varied logic resource utilization across available resources.

Figure 14–1. Average Core Dynamic Power Dissipation

As shown in Figure 14–1, a significant amount of the total power is dissipated in routing for both Stratix III and Cyclone III devices, with the remaining power dissipated in logic, clock, and RAM blocks.

In Stratix and Cyclone device families, a series of column and row interconnect wires of varying lengths provide signal interconnections between logic array blocks (LABs), memory block structures, and digital signal processing (DSP) blocks or multiplier blocks. These interconnects dissipate the largest component of device power.

FPGA combinational logic is another source of power consumption. The basic building block of logic in the latest Stratix series devices is the ALM, and in Cyclone II, Cyclone III and Cyclone IV GX devices, it is the logic element (LE).

For more information about ALMs and LEs in Cyclone II, Cyclone III, Cyclone IV GX, Stratix II, Stratix III, Stratix IV, and Stratix V, devices, refer to the respective device handbook.

Notes to Figure 14–1:
(1) 103 different designs were used to obtain these results.
(2) 96 different designs were used to obtain these results.
(3) In designs using DSP blocks, DSPs consumed 5% of core dynamic power.
Memory and clock resources are other major consumers of power in FPGAs. Stratix II devices feature the TriMatrix memory architecture. TriMatrix memory includes 512-bit M512 blocks, 4-Kbit M4K blocks, and 512-Kbit M-RAM blocks, which are configurable to support many features. Stratix IV and Stratix III TriMatrix on-chip memory is an enhancement based upon the Stratix II FPGA TriMatrix memory and includes three sizes of memory blocks: MLAB blocks, M9K blocks, and M144K blocks. Stratix III, Stratix IV, and Stratix V devices feature Programmable Power Technology, an advanced architecture that enables a smooth trade-off between speed and power. The core of each Stratix III, Stratix IV, and Stratix V device is divided into tiles, each of which may be put into a high-speed or low-power mode. The primary benefit of Programmable Power Technology is to reduce static power, with a secondary benefit being a small reduction in dynamic power. Cyclone II devices have 4-Kbit M4K memory blocks, and Cyclone III and Cyclone IV GX devices have 9-Kbit M9K memory blocks.

Design Space Explorer

Design Space Explorer (DSE) is a simple, easy-to-use, design optimization utility that is included in the Quartus II software. DSE explores and reports optimal Quartus II software options for your design, targeting either power optimization, design performance, or area utilization improvements. You can use DSE to implement the techniques described in this chapter.

Figure 14–2 shows the DSE user interface. The Settings tab is divided into Project Settings and Exploration Settings.
The **Search for Lowest Power** option, under **Exploration Settings**, uses a predefined exploration space that targets overall design power improvements. This setting focuses on applying different options that specifically reduce total design thermal power.

By default, the Quartus II PowerPlay Power Analyzer is run for every exploration performed by the DSE when the **Search for Lowest Power** option is selected. This helps you debug your design and determine trade-offs between power requirements and performance optimization.

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

---

**Power-Driven Compilation**

The standard Quartus II compilation flow consists of Analysis and Synthesis, placement and routing, Assembly, and Timing Analysis. Power-driven compilation takes place at the Analysis and Synthesis and Place-and-Route stages. Quartus II software settings that control power-driven compilation are located in the **PowerPlay power optimization** list on the **Analysis & Synthesis Settings** page, and the **PowerPlay power optimization** list on the **Fitter Settings** page. The following sections describes these power optimization options at the Analysis and Synthesis and Fitter levels.

**Power-Driven Synthesis**

Synthesis netlist optimization occurs during the synthesis stage of the compilation flow. The optimization technique makes changes to the synthesis netlist to optimize your design according to the selection of area, speed, or power optimization. This section describes power optimization techniques at the synthesis level.
The **Analysis & Synthesis Settings** page allows you to specify logic synthesis options. The **PowerPlay power optimization** option is available for all devices supported by the Quartus II software except MAX® 3000 and MAX 7000 devices. (Figure 14–3).

![Figure 14–3. Analysis & Synthesis Settings Page](image)

Table 14–1 shows the settings in the **PowerPlay power optimization** list. You can apply these settings on a project or entity level.

**Table 14–1. Optimize Power During Synthesis Options**

<table>
<thead>
<tr>
<th>Settings</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Off</td>
<td>No netlist, placement, or routing optimizations are performed to minimize power.</td>
</tr>
<tr>
<td>Normal compilation (Default)</td>
<td>Low compute effort algorithms are applied to minimize power through netlist optimizations as long as they are not expected to reduce design performance.</td>
</tr>
<tr>
<td>Extra effort</td>
<td>High compute effort algorithms are applied to minimize power through netlist optimizations. Max performance might be impacted.</td>
</tr>
</tbody>
</table>

The **Normal compilation** setting is turned on by default. This setting performs memory optimization and power-aware logic mapping during synthesis.
Memory blocks can represent a large fraction of total design dynamic power as described in “Reducing Memory Power Consumption” on page 14–14. Minimizing the number of memory blocks accessed during each clock cycle can significantly reduce memory power. Memory optimization involves effective movement of user-defined read/write enable signals to associated read-and-write clock enable signals for all memory types (Figure 14–4).

Figure 14–4 shows a default implementation of a simple dual-port memory block in which write-clock enable signals and read-clock enable signals are connected to VCC, making both read and write memory ports active during each clock cycle. Memory transformation effectively moves the read-enable and write-enable signals to the respective read-clock enable and write-clock enable signals. By using this technique, memory ports are shut down when they are not accessed. This significantly reduces your design’s memory power consumption. For more information about clock enable signals, refer to “Reducing Memory Power Consumption” on page 14–14. For Stratix III, Stratix IV, and Stratix V devices, the memory transformation takes place at the Fitter level by selecting the Normal compilation settings for the power optimization option.

In Stratix III, Cyclone III, Cyclone IV GX, and Stratix III devices, the specified read-during-write behavior can significantly impact the power of single-port and bidirectional dual-port RAMs. It is best to set the read-during-write parameter to “Don’t care” (at the HDL level), as it allows an optimization whereby the read-enable signal can be set to the inversion of the existing write-enable signal (if one exists). This allows the core of the RAM to shut down (that is, not toggle), which saves a significant amount of power.

The other type of power optimization that takes place with the Normal compilation setting is power-aware logic mapping. The power-aware logic mapping reduces power by rearranging the logic during synthesis to eliminate nets with high toggle rates.

The Extra effort setting performs the functions of the Normal compilation setting and other memory optimizations to further reduce memory power by shutting down memory blocks that are not accessed. This level of memory optimization can require extra logic, which can reduce design performance.
The **Extra effort** setting also performs power-aware memory balancing. Power-aware memory balancing automatically chooses the best memory configuration for your memory implementation and provides optimal power saving by determining the number of memory blocks, decoder, and multiplexer circuits required. If you have not previously specified target-embedded memory blocks for your design’s memory functions, the power-aware balancer automatically selects them during memory implementation.

Figure 14–5 shows an example of a 4k × 4 (4k deep and 4 bits wide) memory implementation in two different configurations using M4K memory blocks available in Stratix II devices. The minimum logic area implementation uses M4K blocks configured as 4k × 1. This implementation is the default in the Quartus II software because it has the minimum logic area (0 logic cells) and the highest speed. However, all four M4K blocks are active on each memory access in this implementation, which increases RAM power. The minimum RAM power implementation is created by selecting **Extra effort** in the **PowerPlay power optimization** list. This implementation automatically uses four M4K blocks configured as 1k × 4 for optimal power saving. An address decoder is implemented by the RAM megafunction to select which of the four M4K blocks should be activated on a given cycle, based on the state of the top two user address bits. The RAM megafunction automatically implements a multiplexer to feed the downstream logic by choosing the appropriate M4K output. This implementation reduces RAM power because only one M4K block is active on any cycle, but it requires extra logic cells, costing logic area and potentially impacting design performance.

There is a trade-off between power saved by accessing fewer memories and power consumed by the extra decoder and multiplexor logic. The Quartus II software automatically balances the power savings against the costs to choose the lowest power configuration for each logical RAM. The benchmark data shows that the power-driven synthesis can reduce memory power consumption by as much as 60% in Stratix devices.

---

**Figure 14–5. 4K × 4 Memory Implementation Using Multiple M4K Blocks**

![Diagram showing two configurations of 4K × 4 memory implementation using M4K blocks]
Memory optimization options can also be controlled by the `Low_Power_Mode` parameter in the `Default Parameters` page of the `Settings` dialog box. The settings for this parameter are `None`, `Auto`, and `ALL`. `None` corresponds to the `Off` setting in the `PowerPlay power optimization` list. `Auto` corresponds to the `Normal compilation` setting and `ALL` corresponds to the `Extra effort` setting, respectively. You can apply PowerPlay power optimization either on a compiler basis or on individual entities. The `Low_Power_Mode` parameter always takes precedence over the `Optimize Power for Synthesis` option for power optimization on memory.

You can also set the `MAXIMUM_DEPTH` parameter manually to configure the memory for low power optimization. This technique is the same as the power-aware memory balancer, but it is manual rather than automatic like the `Extra effort` setting in the `PowerPlay power optimization` list. You can set the `MAXIMUM_DEPTH` parameter for memory modules manually in the megafuction instantiation or in the MegaWizard™ Plug-In Manager for power optimization as described in “Reducing Memory Power Consumption” on page 14–14. The `MAXIMUM_DEPTH` parameter always takes precedence over the `Optimize Power for Synthesis` options for power optimization on memory optimization.

For step-by-step instructions on how to perform power-driven synthesis, refer to "Running a Power-Optimized Compilation" in Quartus II Help.
Power-Driven Fitter

The Fitter Settings page enables you to specify options for fitting (Figure 14–6). The PowerPlay power optimization option is available for Arria GX, Arria II GX, Cyclone II, Cyclone III, Cyclone IV, HardCopy series, Stratix II, Stratix II GX, Stratix III, Stratix IV, and Stratix V devices.

Table 14–2 lists the settings in the PowerPlay power optimization list. These settings can only be applied on a project-wide basis. The Extra effort setting for the Fitter requires extensive effort to optimize the design for power and can increase the compilation time.

### Table 14–2. Power-Driven Fitter Option

<table>
<thead>
<tr>
<th>Settings</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Off</td>
<td>No netlist, placement, or routing optimizations are performed to minimize power.</td>
</tr>
<tr>
<td>Normal compilation</td>
<td>Low compute effort algorithms are applied to minimize power through placement and routing optimizations as long as they are not expected to reduce design performance.</td>
</tr>
<tr>
<td>Extra effort</td>
<td>High compute effort algorithms are applied to minimize power through placement and routing optimizations. Max performance might be impacted.</td>
</tr>
</tbody>
</table>
The **Normal compilation** setting is selected by default and performs DSP optimization by creating power-efficient DSP block configurations for your DSP functions. For Stratix III, Stratix IV, and Stratix V devices, this setting, which is based on timing constraints entered for the design, enables the Programmable Power Technology to configure tiles as high-speed mode or low-power mode. Programmable Power Technology is always turned **ON** even when the **OFF** setting is selected for the **Fitter PowerPlay power optimization** option. Tiles are the combination of LAB and MLAB pairs (including the adjacent routing associated with LAB and MLAB), which can be configured to operate in high-speed or low-power mode. This level of power optimization does not have any affect on the fitting, timing results, or compile time. Also, for Stratix III devices, this setting enables the memory transformation as described in “Power-Driven Synthesis” on page 14–4.

For more information about Stratix III power optimization, refer to *AN 437: Power Optimization in Stratix III FPGAs*. For more information about Stratix IV power optimization, refer to *AN 514: Power Optimization in Stratix IV FPGAs*.

The **Extra effort** setting performs the functions of the **Normal compilation** setting and other place-and-route optimizations during fitting to fully optimize the design for power. The Fitter applies an extra effort to minimize power even after timing requirements have been met by effectively moving the logic closer during placement to localize high-toggling nets, and using routes with low capacitance. However, this effort can increase the compilation time.

The **Extra effort** setting uses a Value Change Dump File (`.vcd`) that guides the Fitter to fully optimize the design for power, based on the signal activity of the design. The best power optimization during fitting results from using the most accurate signal activity information. Signal activities from full post-fit netlist (timing) simulation provide the highest accuracy because all node activities reflect the actual design behavior, provided that supplied input vectors are representative of typical design operation. If you do not have a `.vcd` file, the Quartus II software uses assignments, clock assignments, and vectorless estimation values (PowerPlay Power Analyzer Tool settings) to estimate the signal activities. This information is used to optimize your design for power during fitting. The benchmark data shows that the power-driven Fitter technique can reduce power consumption by as much as 19% in Stratix devices. On average, you can reduce core dynamic power by 16% with the Extra effort synthesis and Extra effort fitting settings, as compared to the **Off** settings in both synthesis and Fitter options for power-driven compilation.

Only the **Extra effort** setting in the **PowerPlay power optimization** list for the Fitter option uses the signal activities (from `.vcd` files) during fitting. The settings made in the **PowerPlay Power Analyzer Settings** page in the **Settings** dialog box are used to calculate the signal activity of your design.

For more information about `.vcd` files and how to create them, refer to the **PowerPlay Power Analysis** chapter in volume 3 of the *Quartus II Handbook*.

For step-by-step instructions on how to perform power-driven fitting, refer to *Running a Power-Optimized Compilation* in Quartus II Help.
Area-Driven Synthesis

Using area optimization rather than timing or delay optimization during synthesis saves power because you use fewer logic blocks. Using less logic usually means less switching activity. The Quartus II integrated synthesis tool provides Speed, Balanced, or Area for the Optimization Technique option. You can also specify this logic option for specific modules in your design with the Assignment Editor in cases where you want to reduce area using the Area setting (potentially at the expense of register-to-register timing performance) while leaving the default Optimization Technique setting at Balanced (for the best trade-off between area and speed for certain device families). The Speed Optimization Technique can increase the resource usage of your design if the constraints are too aggressive, and can also result in increased power consumption.

The benchmark data shows that the area-driven technique can reduce power consumption by as much as 31% in Stratix devices and as much as 15% in Cyclone devices.

Gate-Level Register Retiming

You can also use gate-level register retiming to reduce circuit switching activity. Retiming shuffles registers across combinational blocks without changing design functionality. The Perform gate-level register retiming option in the Quartus II software enables the movement of registers across combinational logic to balance timing, allowing the software to trade off the delay between timing critical and noncritical timing paths.

Retiming uses fewer registers than pipelining. Figure 14–7 shows an example of gate-level register retiming, where the 10 ns critical delay is reduced by moving the register relative to the combinational logic, resulting in the reduction of data depth and switching activity.

Figure 14–7. Gate-Level Register Retiming
Gate-level register retiming makes changes at the gate level. If you are using an atom netlist from a third-party synthesis tool, you must also select the Perform WYSIWYG primitive resynthesis option to undo the atom primitives to gates mapping (so that register retiming can be performed), and then to remap gates to Altera primitives. When using Quartus II integrated synthesis, retiming occurs during synthesis before the design is mapped to Altera primitives. The benchmark data shows that the combination of WYSIWYG remapping and gate-level register retiming techniques can reduce power consumption by as much as 6% in Stratix devices and as much as 21% in Cyclone devices.

For more information about register retiming, refer to the Netlist Optimizations and Physical Synthesis chapter in volume 2 of the Quartus II Handbook.

Design Guidelines

Several low-power design techniques can reduce power consumption when applied during FPGA design implementation. This section provides detailed design techniques for Cyclone II, Cyclone III, Cyclone IV GX, Stratix II, and Stratix III devices that affect overall design power. The results of these techniques might be different from design to design.

Clock Power Management

Clocks represent a significant portion of dynamic power consumption due to their high switching activity and long paths. Figure 14-1 on page 14-2 shows a 14% average contribution to power consumption for global clock routing in Stratix III devices and 16% in Cyclone III devices. Actual clock-related power consumption is higher than this because the power consumed by local clock distribution within logic, memory, and DSP or multiplier blocks is included in the power consumption for the respective blocks.

Clock routing power is automatically optimized by the Quartus II software, which enables only those portions of the clock network that are required to feed downstream registers. Power can be further reduced by gating clocks when they are not required. It is possible to build clock-gating logic, but this approach is not recommended because it is difficult to generate a glitch free clock in FPGAs using ALMs or LEs.

Arria GX, Arria II GX, Cyclone III, Cyclone IV, Stratix II, Stratix III, Stratix IV, and Stratix V devices use clock control blocks that include an enable signal. A clock control block is a clock buffer that lets you dynamically enable or disable the clock network and dynamically switch between multiple sources to drive the clock network. You can use the Quartus II MegaWizard Plug-In Manager to create this clock control block with the ALTCLKCTRL megafunction. Arria GX, Arria II GX, Cyclone III, Cyclone IV, Stratix II, Stratix III, Stratix IV, and Stratix V devices provide clock control blocks for global clock networks. In addition, Stratix II, Stratix III,
Stratix IV, and Stratix V devices have clock control blocks for regional clock networks. The dynamic clock enable feature lets internal logic control the clock network. When a clock network is powered down, all the logic fed by that clock network does not toggle, thereby reducing the overall power consumption of the device. Figure 14–8 shows a 4-input clock control block diagram.

**Figure 14–8. Clock Control Block Diagram**

The enable signal is applied to the clock signal before being distributed to global routing. Therefore, the enable signal can either have a significant timing slack (at least as large as the global routing delay) or it can reduce the $f_{\text{MAX}}$ of the clock signal.

For more information about using clock control blocks, refer to the *Clock Control Block Megafunction User Guide (ALTCLKCTRL)*.

Another contributor to clock power consumption is the LAB clock that distributes a clock to the registers within a LAB. LAB clock power can be the dominant contributor to overall clock power. For example, in Cyclone III devices, each LAB can use two clocks and two clock enable signals, as shown in Figure 14–9. Each LAB’s clock signal and clock enable signal are linked. For example, an LE in a particular LAB using the $\text{labclk}_1$ signal also uses the $\text{labclkena}_1$ signal.

**Figure 14–9. LAB-Wide Control Signals**
To reduce LAB-wide clock power consumption without disabling the entire clock tree, use the LAB-wide clock enable to gate the LAB-wide clock. The Quartus II software automatically promotes register-level clock enable signals to the LAB-level. All registers within an LAB that share a common clock and clock enable are controlled by a shared gated clock. To take advantage of these clock enables, use a clock enable construct in the relevant HDL code for the registered logic.

**LAB-Wide Clock Enable Example**

The VHDL code in Example 14–1 makes use of a LAB-wide clock enable. This clock-gating logic is automatically turned into an LAB-level clock enable signal.

```
IF clk'event AND clock = '1' THEN
  IF logic_is_enabled = '1' THEN
    reg <= value;
  ELSE
    reg <= reg;
  END IF;
END IF;
```

For more information about LAB-wide control signals, refer to the *Stratix II Architecture, Cyclone III Device Family Overview,* or *Cyclone II Architecture* chapters in the respective device handbook.

**Reducing Memory Power Consumption**

The memory blocks in FPGA devices can represent a large fraction of typical core dynamic power. Memory consumes approximately 20% of the core dynamic power in typical Cyclone III and Stratix III device designs. Memory blocks are unlike most other blocks in the device because most of their power is tied to the clock rate, and is insensitive to the toggle rate on the data and address lines.

When a memory block is clocked, there is a sequence of timed events that occur within the block to execute a read or write. The circuitry controlled by the clock consumes the same amount of power regardless of whether or not the address or data has changed from one cycle to the next. Thus, the toggle rate of input data and the address bus have no impact on memory power consumption.

The key to reducing memory power consumption is to reduce the number of memory clocking events. You can achieve this through clock network-wide gating described in “Clock Power Management” on page 14–12, or on a per-memory basis through use of the clock enable signals on the memory ports. Figure 14–10 shows the logical view of the internal clock of the memory block. Use the appropriate enable signals on the memory to make use of the clock enable signal instead of gating the clock.

**Figure 14–10. Memory Clock Enable Signal**
Using the clock enable signal enables the memory only when necessary and shuts it down for the rest of the time, reducing the overall memory power consumption. You can use the MegaWizard Plug-In Manager to create these enable signals by selecting the Clock enable signal option for the appropriate port when generating the memory block function (Figure 14–11).

![Figure 14–11. MegaWizard Plug-In Manager RAM 2-Port Clock Enable Signal Selectable Option](image)

For example, consider a design that contains a 32-bit-wide M4K memory block in ROM mode that is running at 200 MHz. Assuming that the output of this block is only required approximately every four cycles, this memory block will consume 8.45 mW of dynamic power according to the demands of the downstream logic. By adding a small amount of control logic to generate a read clock enable signal for the memory block only on the relevant cycles, the power can be cut 75% to 2.15 mW.

You can also use the MAXIMUM_DEPTH parameter in your memory megafunction to save power in Cyclone II, Cyclone III, Cyclone IV GX, Stratix II, Stratix III, Stratix IV, and Stratix V devices; however, this approach might increase the number of LEs required to implement the memory and affect design performance.

You can set the MAXIMUM_DEPTH parameter for memory modules manually in the megafunction instantiation or in the MegaWizard Plug-In Manager (Figure 14–12). The Quartus II software automatically chooses the best design memory configuration for optimal power, as described in “Power-Driven Compilation” on page 14–4.
Memory Power Reduction Example

Table 14–3 shows power usage measurements for a 4K × 36 simple dual-port memory implemented using multiple M4K blocks in a Stratix II EP2S15 device. For each implementation, the M4K blocks are configured with a different memory depth.

Table 14–3. 4K × 36 Simple Dual-Port Memory Implemented Using Multiple M4K Blocks

<table>
<thead>
<tr>
<th>M4K Configuration</th>
<th>Number of M4K Blocks</th>
<th>ALUTs</th>
</tr>
</thead>
<tbody>
<tr>
<td>4K × 1 (Default setting)</td>
<td>36</td>
<td>0</td>
</tr>
<tr>
<td>2K × 2</td>
<td>36</td>
<td>40</td>
</tr>
<tr>
<td>1K × 4</td>
<td>36</td>
<td>62</td>
</tr>
<tr>
<td>512 × 9</td>
<td>32</td>
<td>143</td>
</tr>
<tr>
<td>256 × 18</td>
<td>32</td>
<td>302</td>
</tr>
<tr>
<td>128 × 36</td>
<td>32</td>
<td>633</td>
</tr>
</tbody>
</table>
Figure 14–13 shows the amount of power saved using the MAXIMUM DEPTH parameter. For all implementations, a user-provided read enable signal is present to indicate when read data is required. Using this power-saving technique can reduce power consumption by as much as 60%.

As the memory depth becomes more shallow, memory dynamic power decreases because unaddressed M4K blocks can be shut off using a decoded combination of address bits and the read enable signal. For a 128-deep memory block, power used by the extra LEs starts to outweigh the power gain achieved by using a more shallow memory block depth. The power consumption of the memory blocks and associated LEs depends on the memory configuration.

**Pipelining and Retiming**

Designs with many glitches consume more power because of faster switching activity. Glitches cause unnecessary and unpredictable temporary logic switches at the output of combinational logic. A glitch usually occurs when there is a mismatch in input signal timing leading to unequal propagation delay.

For example, consider an input change on one input of a 2-input XOR gate from 1 to 0, followed a few moments later by an input change from 0 to 1 on the other input. For a moment, both inputs become 1 (high) during the state transition, resulting in 0 (low) at the output of the XOR gate. Subsequently, when the second input transition takes place, the XOR gate output becomes 1 (high). During signal transition, a glitch is produced before the output becomes stable, as shown in Figure 14–14. This glitch can propagate to subsequent logic and create unnecessary switching activity, increasing power consumption. Circuits with many XOR functions, such as arithmetic circuits or cyclic redundancy check (CRC) circuits, tend to have many glitches if there are several levels of combinational logic between registers.
Pipelining can reduce design glitches by inserting flipflops into long combinational paths. Flipflops do not allow glitches to propagate through combinational paths. Therefore, a pipelined circuit tends to have less glitching. Pipelining has the additional benefit of generally allowing higher clock speed operations, although it does increase the latency of a circuit (in terms of the number of clock cycles to a first result). Figure 14–15 shows an example where pipelining is applied to break up a long combinational path.

**Figure 14–15. Pipelining Example**

![Pipelining Example](image)

Pipelining is very effective for glitch-prone arithmetic systems because it reduces switching activity, resulting in reduced power dissipation in combinational logic. Additionally, pipelining allows higher-speed operation by reducing logic-level numbers between registers. The disadvantage of this technique is that if there are not many glitches in your design, pipelining can increase power consumption by adding unnecessary registers. Pipelining can also increase resource utilization. The benchmark data shows that pipelining can reduce dynamic power consumption by as much as 30% in Cyclone and Stratix devices.

**Architectural Optimization**

You can use design-level architectural optimization by taking advantage of specific device architecture features. These features include dedicated memory and DSP or multiplier blocks available in FPGA devices to perform memory or arithmetic-related functions. You can use these blocks in place of LUTs to reduce power consumption. For example, you can build large shift registers from RAM-based FIFO buffers instead of building the shift registers from the LE registers.

The Stratix device family allows you to efficiently target small, medium, and large memories with the TriMatrix memory architecture. Each TriMatrix memory block is optimized for a specific function. The M512 memory blocks available in Stratix II devices are useful for implementing small FIFO buffers, DSP, and clock domain transfer applications. M512 memory blocks are more power-efficient than the distributed memory structures in some competing FPGAs. The M4K memory blocks...
are used to implement buffers for a wide variety of applications, including processor code storage, large look-up table implementation, and large memory applications. The M-RAM blocks are useful in applications where a large volume of data must be stored on-chip. Effective utilization of these memory blocks can have a significant impact on power reduction in your design.

The latest Stratix and Cyclone device families have configurable M9K memory blocks that provide various memory functions such as RAM, FIFO buffers, and ROM.

For more information about using DSP and memory blocks efficiently, refer to the 
*Area and Timing Optimization* chapter in volume 2 of the *Quartus II Handbook*.

**I/O Power Guidelines**

Nonterminated I/O standards such as LVTTL and LVCMOS have a rail-to-rail output swing. The voltage difference between logic-high and logic-low signals at the output pin is equal to the $V_{CCIO}$ supply voltage. If the capacitive loading at the output pin is known, the dynamic power consumed in the I/O buffer can be calculated as shown in Equation 14–1:

\[
P = 0.5 \times F \times C \times V^2
\]

In this equation, $F$ is the output transition frequency and $C$ is the total load capacitance being switched. $V$ is equal to $V_{CCIO}$ supply voltage. Because of the quadratic dependence on $V_{CCIO}$, lower voltage standards consume significantly less dynamic power.

Transistor-to-transistor logic (TTL) I/O buffers consume very little static power. As a result, the total power consumed by a LVTTL or LVCMOS output is highly dependent on load and switching frequency.

When using resistively terminated I/O standards like SSTL and HSTL, the output load voltage swings by a small amount around some bias point. The same dynamic power equation is used, where $V$ is the actual load voltage swing. Because this is much smaller than $V_{CCIO}$, dynamic power is lower than for nonterminated I/O under similar conditions. These resistively terminated I/O standards dissipate significant static (frequency-independent) power, because the I/O buffer is constantly driving current into the resistive termination network. However, the lower dynamic power of these I/O standards means they often have lower total power than LVCMOS or LVTTL for high-frequency applications. Use the lowest drive strength I/O setting that meets your speed and waveform requirements to minimize I/O power when using resistively terminated standards.

You can save a small amount of static power by connecting unused I/O banks to the lowest possible $V_{CCIO}$ voltage of 1.2 V.

*Table 14–4* shows the total supply and thermal power consumed by outputs using different I/O standards for Stratix II devices. The numbers are for an I/O pin transmitting random data clocked at 200 MHz with a 10 pF capacitive load.
For this configuration, nonterminated standards generally use less power, but this is not always the case. If the frequency or the capacitive load is increased, the power consumed by nonterminated outputs increases faster than the power of terminated outputs.

### Table 14–4. I/O Power for Different I/O Standards in Stratix II Devices

<table>
<thead>
<tr>
<th>Standard</th>
<th>Total Supply Current Drawn from $V_{CCIO}$ Supply (mA)</th>
<th>Total On-Chip Thermal Power Dissipation (mW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3-V LVTTL</td>
<td>2.42</td>
<td>9.87</td>
</tr>
<tr>
<td>2.5-V LVCMOS</td>
<td>1.9</td>
<td>6.69</td>
</tr>
<tr>
<td>1.8-V LVCMOS</td>
<td>1.34</td>
<td>4.18</td>
</tr>
<tr>
<td>1.5-V LVCMOS</td>
<td>1.18</td>
<td>3.58</td>
</tr>
<tr>
<td>3.3-V PCI</td>
<td>2.47</td>
<td>10.23</td>
</tr>
<tr>
<td>SSTL-2 class I</td>
<td>6.07</td>
<td>4.42</td>
</tr>
<tr>
<td>SSTL-2 class II</td>
<td>10.72</td>
<td>5.1</td>
</tr>
<tr>
<td>SSTL-18 class I</td>
<td>5.33</td>
<td>3.28</td>
</tr>
<tr>
<td>SSTL-18 class II</td>
<td>8.56</td>
<td>4.06</td>
</tr>
<tr>
<td>HSTL-15 class I</td>
<td>6.06</td>
<td>3.49</td>
</tr>
<tr>
<td>HSTL-15 class II</td>
<td>11.08</td>
<td>4.87</td>
</tr>
<tr>
<td>HSTL-18 class I</td>
<td>6.87</td>
<td>4.09</td>
</tr>
<tr>
<td>HSTL-18 class II</td>
<td>12.33</td>
<td>5.82</td>
</tr>
</tbody>
</table>


When calculating I/O power, the PowerPlay Power Analyzer uses the default capacitive load set for the I/O standard in the Capacitive Loading page of the Device and Pin Options dialog box. For Stratix II devices, if Enable Advanced I/O Timing is turned on, I/O power is measured using an equivalent load calculated as the sum of the near capacitance, the transmission line distributed capacitance, and the far-end capacitance as defined in the Board Trace Model page of the Device and Pin Options dialog box or the Board Trace Model view in the Pin Planner. Any other components defined in the board trace model are not taken into account for the power measurement.

For Cyclone III, Cyclone IV GX, Stratix III, Stratix IV, and Stratix V, devices, Advanced I/O Timing, which uses the full board trace model, is always used.

For information about using Advanced I/O Timing and configuring a board trace model, refer to the I/O Management chapter in volume 2 of the Quartus II Handbook.
Dynamically Controlled On-Chip Terminations

Stratix V, Stratix IV and Stratix III FPGAs offer dynamic on-chip termination (OCT). Dynamic OCT enables series termination (RS) and parallel termination (RT) to dynamically turn on/off during the data transfer. This feature is especially useful when Stratix V, Stratix IV and Stratix III FPGAs are used with external memory interfaces, such as interfacing with DDR memories.

Compared to conventional termination, dynamic OCT reduces power consumption significantly as it eliminates the constant DC power consumed by parallel termination when transmitting data. Parallel termination is extremely useful for applications that interface with external memories where I/O standards, such as HSTL and SSTL, are used. Parallel termination supports dynamic OCT, which is useful for bidirectional interfaces (see Figure 14–16).

The following is an example of power saving for a DDR3 interface using on-chip parallel termination.

The static current consumed by parallel OCT is equal to the $V_{CCIO}$ voltage divided by 100 Ω. For DDR3 interfaces that use SSTL-15, the static current is $1.5 \text{ V} / 100 \Omega = 15 \text{ mA}$ per pin. Therefore, the static power is $1.5 \text{ V} \times 15 \text{ mA} = 22.5 \text{ mW}$. For an interface with 72 DQ and 18 DQS pins, the static power is $90 \text{ pins} \times 22.5 \text{ mW} = 2.025 \text{ W}$. Dynamic parallel OCT disables parallel termination during write operations, so if writing occurs 50% of the time, the power saved by dynamic parallel OCT is $50\% \times 2.025 \text{ W} = 1.0125 \text{ W}$.

For more information about dynamic OCT in Stratix IV and Stratix III devices, refer to the Stratix III Device I/O Features chapter in the Stratix III Device Handbook and the Stratix IV Device I/O Features chapter in the Stratix IV Device Handbook, respectively.

**Power Optimization Advisor**

The Quartus II software includes the Power Optimization Advisor, which provides specific power optimization advice and recommendations based on the current design project settings and assignments. The advisor covers many of the suggestions listed in this chapter. The following example shows how to reduce your design power with the Power Optimization Advisor.
Power Optimization Advisor Example

After compiling your design, run the PowerPlay Power Analyzer to determine your design power and to see where power is dissipated in your design. Based on this information, you can run the Power Optimization Advisor to implement recommendations that can reduce design power. Figure 14–17 shows the Power Optimization Advisor after compiling a design that is not fully optimized for power.

Figure 14–17. Power Optimization Advisor

The Power Optimization Advisor shows the recommendations that can reduce power in your design. The recommendations are split into stages to show the order in which you should apply the recommended settings. The first stage shows mostly CAD setting options that are easy to implement and highly effective in reducing design power. An icon indicates whether each recommended setting is made in the current project. In Figure 14–17, the checkmark icons for Stage 1 shows the recommendations that are already implemented. The warning icons indicate recommendations that are not followed for this compilation. The information icon shows the general suggestions. Each recommendation includes the description, summary of the effect of the recommendation, and the action required to make the appropriate setting.
There is a link from each recommendation to the appropriate location in the Quartus II user interface where you can change the setting. You can change the Power-Driven Synthesis setting by clicking Open Settings dialog box - Analysis & Synthesis Settings page (Figure 14–18). The Settings dialog box is shown with the Analysis & Synthesis Settings page selected, where you can change the PowerPlay power optimization settings.

Figure 14–18. Analysis & Synthesis Settings Page
After making the recommended changes, recompile your design. The Power Optimization Advisor indicates with green check marks that the recommendations were implemented successfully (Figure 14–19). You can use the PowerPlay Power Analyzer to verify your design power results.

Figure 14–19. Implementation of Power Optimization Advisor Recommendations

The recommendations listed in Stage 2 generally involve design changes, rather than CAD settings changes as in Stage 1. You can use these recommendations to further reduce your design power consumption. Altera recommends that you implement Stage 1 recommendations first, then the Stage 2 recommendations.

Conclusion

The combination of a smaller process technology, the use of low-k dielectric material, and reduced supply voltage significantly reduces dynamic power consumption in the latest FPGAs. To further reduce your dynamic power, use the design recommendations presented in this chapter to optimize resource utilization and minimize power consumption.

Document Revision History

Table 14–5 shows the revision history for this chapter.

Table 14–5. Document Revision History (Part 1 of 2)

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>December 2010</td>
<td>10.0.1</td>
<td>Template update.</td>
</tr>
<tr>
<td>July 2010</td>
<td>10.0.0</td>
<td>-Was chapter 11 in the 9.1.0 release</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-Updated Figures 14-2, 14-3, 14-6, 14-18, 14-19, and 14-20</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-Updated device support</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-Minor editorial updates</td>
</tr>
</tbody>
</table>
### Table 14–5. Document Revision History (Part 2 of 2)

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>November 2009</td>
<td>9.1.0</td>
<td>• Updated Figure 11-1 and associated references</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Updated device support</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Minor editorial update</td>
</tr>
<tr>
<td>March 2009</td>
<td>9.0.0</td>
<td>• Was chapter 9 in the 8.1.0 release</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Updated for the Quartus II software release</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added benchmark results</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Removed several sections</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Updated Figure 14–1, Figure 14–17, Figure 14–18, and Figure 14–19</td>
</tr>
<tr>
<td>November 2008</td>
<td>8.1.0</td>
<td>• Changed to 8½” × 11” page size</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Changed references to altsyncram to RAM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Minor editorial updates</td>
</tr>
<tr>
<td>May 2008</td>
<td>8.0.0</td>
<td>• Added support for Stratix IV devices</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Updated Table 9–1 and 9–9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Updated “Architectural Optimization” on page 9–22</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added “Dynamically-Controlled On-Chip Terminations” on page 9–26</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Updated “Referenced Documents” on page 9–29</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Updated references</td>
</tr>
</tbody>
</table>

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

* Take an online survey to provide feedback about this handbook chapter.
As FPGA designs grow larger in density, the ability to analyze the design for performance, routing congestion, and logic placement to meet the design requirements becomes critical. This chapter discusses how to analyze the design floorplan with the Chip Planner.

You can perform design analysis and create and optimize the design floorplan with the Chip Planner. To make I/O assignments, use the Pin Planner.

For information about the Pin Planner, refer to the I/O Management chapter in volume 2 of the Quartus II Handbook.

You can use the Design Partition Planner with the Chip Planner to customize the floorplan of your design. For more information, refer to the Quartus II Incremental Compilation for Hierarchical and Team-Based Design and the Best Practices for Incremental Compilation Partitions and Floorplan Assignments chapters in volume 1 of the Quartus II Handbook.

This chapter includes the following topics:
- “Chip Planner Overview”
- “LogicLock Regions” on page 15–3
- “Using LogicLock Regions in the Chip Planner” on page 15–10
- “Design Floorplan Analysis Using the Chip Planner” on page 15–11
- “Scripting Support” on page 15–20

For a list of devices supported by the Chip Planner, refer to About the Chip Planner in Quartus II Help.

For more information about the Chip Planner, refer to the Altera Training page of the Altera website.

Chip Planner Overview

The Chip Planner provides a visual display of chip resources. The Chip Planner can show logic placement, LogicLock regions, relative resource usage, detailed routing information, fan-in and fan-out connections between nodes, timing paths between registers, delay estimates for paths, and routing congestion information.
You can also make assignment changes with the Chip Planner, such as creating and deleting resource assignments, and you can perform post-compilation changes such as creating, moving, and deleting logic cells and I/O atoms. With the Chip Planner, you can view and create assignments for a design floorplan, perform power and design analyses, and implement ECOs. With the Chip Planner and Resource Property Editor, you can change connections between resources and make post-compilation changes to the properties of logic cells, I/O elements, PLLs, and RAM and digital signal processing (DSP) blocks.

For details about how to implement ECOs in your design using the Chip Planner in the Quartus II software, refer to the Engineering Change Management with the Chip Planner chapter in volume 2 of the Quartus II Handbook.

Starting the Chip Planner

To start the Chip Planner, on the Tools menu, click Chip Planner (Floorplan & Chip Editor). You can also start the Chip Planner by the following methods:

- Click the Chip Planner icon on the Quartus II software toolbar
- On the Shortcut menu in the following tools, click Locate and then click Locate in Chip Planner (Floorplan and Chip Editor):
  - Design Partition Planner
  - Compilation Report
  - LogicLock Regions window
  - Technology Map Viewer
  - Project Navigator window
  - RTL source code
  - Node Finder
  - Simulation Report
  - RTL Viewer
  - Report Timing panel of the TimeQuest Timing Analyzer

Chip Planner Toolbar

The Chip Planner provides powerful tools for design analysis with a GUI. You can access Chip Planner commands from the View menu and the Shortcut menu, or by clicking the icons on the toolbar.

Chip Planner Tasks, Layers, and Editing Modes

The Chip Planner models types of resource objects as unique display layers, and uses tasks—which are predefined sets of layer settings—to control the display of resources. The Chip Planner provides a set of default tasks, and you can create custom tasks to customize the display for your particular needs. The Basic, Detailed, and Floorplan Editing tasks provided with the Chip Planner are useful for general ECO and assignment-related activities, while the Partition Planner, Power, and Routing Congestion tasks are optimized for specific activities.
The Chip Planner has two editing modes, which determine the types of operations that you can perform. The Assignment editing mode allows you to make assignment changes that are applied by the Fitter during the next place and route operation. The ECO editing mode allows you to make post-compilation changes, commonly referred to as engineering change orders (ECOs).

You should choose the editing mode appropriate for the work that you want to perform, and a task that displays the resources that you want to view, in a level of detail appropriate for your design.

**Locate History Window**

As you optimize your design floorplan, you might have to locate a path or node in the Chip Planner many times. The Locate History window lists all the nodes and paths you have displayed using a Locate in Chip Planner (Floorplan and Chip Editor) command, providing easy access to the nodes and paths of interest to you. If you locate a required path from the TimeQuest Timing Analyzer Report Timing pane, the Locate History window displays the required clock path. If you locate an arrival path from the TimeQuest Timing Analyzer Report Timing pane, the Locate History window displays the path from the arrival clock to the arrival data. Double-clicking a node or path in the Locate History window displays the selected node or path in the Chip Planner.

For more information about the Chip Planner, refer to About the Chip Planner and Layers Settings Dialog Box in Quartus II Help. For more information about the ECO editing mode, refer to the Engineering Change Management with the Chip Planner chapter in volume 2 of the Quartus II Handbook.

**LogicLock Regions**

LogicLock regions are floorplan location constraints that help you place logic on the target device. When you assign entity instances or nodes to a LogicLock region, you direct the Fitter to place those entity instances or nodes within the region during fitting. Your floorplan can contain several LogicLock regions.

A LogicLock region is defined by its height, width, and location; you can specify the size or location of a region, or both, or the Quartus II software can generate these properties automatically. The Quartus II software bases the size and location of a region on the contents of the region and the timing requirements of the module. Table 15–1 describes the options for creating LogicLock regions.

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>State</td>
<td>Floating (1), Locked</td>
<td>Floating allows the Quartus II software to determine the location of the region on the device. Floating regions are shown with a dashed boundary in the floorplan. Locked allows you to specify the location of the region. Locked regions are shown with a solid boundary in the floorplan. A locked region must have a fixed size.</td>
</tr>
<tr>
<td>Size</td>
<td>Auto (1), Fixed</td>
<td>Allows the Quartus II software to determine the appropriate size of a region given its contents. Fixed regions have a shape and size that you define.</td>
</tr>
</tbody>
</table>
The Quartus II software cannot automatically define the size of a region if the location is locked. Therefore, if you want to specify the exact location of the region, you must also specify the size.

You can use the Design Partition Planner in conjunction with LogicLock regions to create a floorplan for your design. For more information about using the Design Partition Planner, refer to the Quartus II Incremental Compilation for Hierarchical and Team-Based Designs and the Best Practices for Incremental Compilation Partition and Floorplan Assignments chapters in volume 1 of the Quartus II Handbook.

### Creating LogicLock Regions

You can create LogicLock Regions with the Project Navigator, the LogicLock Regions window, the Design Partition Planner, the Chip Planner, and with Tcl commands.

#### Creating LogicLock Regions with the Project Navigator

After you perform either a full compilation or analysis and elaboration on the design, the Quartus II software displays the hierarchy of the design. On the View menu, click Project Navigator. With the hierarchy of the design fully expanded, right-click on any design entity in the design, and click Create New LogicLock Region to create a LogicLock region and assign the entity to the new region.

#### Creating LogicLock Regions with the LogicLock Regions window

To create a LogicLock region with the LogicLock Regions window, on the Assignments menu, click LogicLock Regions Window. In the LogicLock Regions window, click <<new>>.

#### Creating LogicLock Regions with the Design Partition Planner

To create a LogicLock region and assign a partition to it with the Design Partition Planner, right-click the partition and then click Create LogicLock Region.

#### Creating LogicLock Regions with the Chip Planner

To create a LogicLock region in the Chip Planner, click the Create LogicLock Region command on the View menu, then click and drag on the Chip Planner floorplan to create a region of your preferred location and size.

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reserved</td>
<td>Off (1), On</td>
<td>Allows you to define whether the Fitter can use the resources within a region for entities that are not assigned to the region. If the reserved property is turned on, only items assigned to the region can be placed within its boundaries.</td>
</tr>
<tr>
<td>Origin</td>
<td>Any Floorplan Location</td>
<td>Specifies the location of the LogicLock region on the floorplan. For Arria series, Stratix, Cyclone series, MAX II, and MAX V devices, the origin is located in the lower left corner of the LogicLock region. For other Altera® device families, the origin is located in the upper left corner of the LogicLock region.</td>
</tr>
</tbody>
</table>

**Note to Table 15–1:**

(1) Default value.

Table 15–1. Types of LogicLock Regions (Part 2 of 2)
Creating Nonrectangular LogicLock Regions

When you create a floorplan for your design, you may want to create nonrectangular LogicLock regions to exclude certain resources from the LogicLock region. You might also create a nonrectangular LogicLock region to place certain parts of your design around specific device resources to improve performance.

To create a nonrectangular region with the Merge LogicLock Region command, follow these steps:

1. In the Chip Planner, create two or more contiguous or non-contiguous rectangular regions as described in “Creating LogicLock Regions” on page 15–4.
2. Arrange the regions that you have created into the locations where you want the nonrectangular region to be.
3. Select all the individual regions that you want to merge by clicking each of them while pressing the Shift key.
4. Right-click the title bar of any of the LogicLock regions that you want to merge, point to LogicLock Regions, and then click Merge LogicLock Region. The individual regions that you select merge to create a single new region.

By default, the new LogicLock region has the same name as the component region containing the greatest number of resources; however, you can rename the new region. In the LogicLock Regions Window, the new region is shown as having a Custom Shape.

Figure 15–1 illustrates using the Merge LogicLock Region command to form a nonrectangular LogicLock region by merging two rectangular LogicLock regions.

Figure 15–1. Using the Merge LogicLock Region command to create a nonrectangular region
Hierarchical (Parent and Child) LogicLock Regions

To further constrain module locations, you can define a hierarchy for a group of regions by declaring parent and child regions. The Quartus II software places a child region completely within the boundaries of its parent region; a child region must be placed entirely within the boundary of its parent. Additionally, parent and child regions allow you to further improve the performance of a module by constraining nodes in the critical path of a module.

To make one LogicLock region a child of another LogicLock region, in the LogicLock Regions window, select the new child region and drag and drop the new child region into its new parent region.

The LogicLock region hierarchy does not have to be the same as the design hierarchy.

You can create both auto-sized and fixed-sized LogicLock regions within a parent LogicLock region; however, the parent of a fixed-sized child region must also be fixed-sized. The location of a locked parent region is locked relative to the device; the location of a locked child region is locked relative to its parent region. If you change the parent’s location, the locked child’s origin changes, but maintains the same placement relative to the origin of its parent. The location of a floating child region can float within its parent. Complex region hierarchies might result in some LABs not being used, effectively increasing the resource utilization in the device. Do not create more levels of hierarchy than you need.

Placing LogicLock Regions

A fixed region must contain all resources required by the design block assigned to the region. Although the Quartus II software can automatically place and size LogicLock regions to meet resource and timing requirements, you can manually place and size regions to meet your design requirements. You should consider the following if you manually place or size a LogicLock region:

- LogicLock regions with pin assignments must be placed on the periphery of the device, adjacent to the pins. For the Arria series, Cyclone series, MAX II, MAX V, and Stratix series of devices, you must also include the I/O block within the LogicLock Region.

- Floating LogicLock regions can overlap with their ancestors or descendants, but not with other floating LogicLock regions.

Placing Device Resources into LogicLock Regions

A LogicLock region includes all device resources within its boundaries, including memory and pins. The Quartus II software does not include pins automatically when you assign an entity to a region—you can manually assign pins to LogicLock regions; however, this placement puts location constraints on the region. The software only obeys pin assignments to locked regions that border the periphery of the device. For the Arria series, Cyclone series, MAX II, MAX V, and Stratix series of devices, the locked regions must include the I/O pins as resources.

Pin assignments to LogicLock regions are effective only in fixed and locked regions. Pin assignments to floating regions do not influence the placement of the region.
Only one LogicLock region can claim a device resource. If a LogicLock region boundary includes part of a device resource, the Quartus II software allocates the entire resource to that LogicLock region. When the Quartus II software places a floating auto-sized region, it places the region in an area that meets the requirements of the contents of the LogicLock region.

If you want to import multiple instances of a module into a top-level design, you must ensure that the device has two or more locations with exactly the same device resources. (You can determine this from the applicable device handbook.) If the device does not have another area with exactly the same resources, the Quartus II software generates a fitting error during compilation of the top-level design.

**LogicLock Regions Window**

You can use the LogicLock Regions window to create LogicLock regions, assign nodes and entities to them, and modify the properties of a LogicLock region such as size, state, width, height, origin, and whether the region is a reserved region. The LogicLock Regions window also has a recommendations toolbar; select a LogicLock region from the drop-down list in the recommendations toolbar to display the relevant suggestions to optimize that LogicLock region. You can customize the LogicLock Regions window by dragging and dropping the columns to change their order; you can also show and hide optional columns by right-clicking any column heading and then selecting the appropriate columns in the shortcut menu.

![Figure 15–2. LogicLock Regions Window](image)

The **LogicLock Region Properties** dialog box provides a summary of all LogicLock regions in your design. Use the **LogicLock Region Properties** dialog box to obtain detailed information about your LogicLock region, such as which entities and nodes are assigned to your region and which resources are required. The **LogicLock Region Properties** dialog box shows the properties of the current selected regions and allows you to modify them. To open the **LogicLock Region Properties** dialog box, double-click any region in the LogicLock Regions window, or right-click the region and click **Properties**.

For designs that target Arria series, Cyclone series, Stratix series, MAX II, and MAX V devices, the Quartus II software automatically creates a LogicLock region that encompasses the entire device. This default region is labelled **Root Region**, and is locked and fixed.
For Arria series, Cyclone series, Stratix series, MAX II, and MAX V devices, the origin of the LogicLock region is located at the lower-left corner of the region. For all other supported devices, the origin is located at the upper-left corner of the region.

Reserved LogicLock Region

The Quartus II software honors all entity and node assignments to LogicLock regions. Occasionally, entities and nodes do not occupy an entire region, which leaves some of the region’s resources unoccupied. To increase the region’s resource utilization and performance, the Quartus II software’s default behavior fills the unoccupied resources with other nodes and entities that have not been assigned to another region. You can prevent this behavior by turning on Reserved on the General tab of the LogicLock Region Properties dialog box. When you turn on this option, your LogicLock region contains only the entities and nodes that you specifically assigned to your LogicLock region.

Excluded Resources

The Excluded Resources feature allows you to easily exclude specific device resources such as DSP blocks or M4K memory blocks from a LogicLock region. For example, you can assign a specific entity to a LogicLock region but allow the DSP blocks of that entity to be placed anywhere on the device. Use the Excluded Resources feature on a per-LogicLock region member basis.

To exclude certain device resources from an entity, in the LogicLock Region Properties dialog box, highlight the entity in the Design Element column, and click Edit. In the Edit Node dialog box, under Excluded Element Types, click the Browse button. In the Excluded Resources Element Types dialog box, you can select the device resources you want to exclude from the entity. When you have selected the resources to exclude, the Excluded Resources column is updated in the LogicLock Region Properties dialog box to reflect the excluded resources.

The Excluded Resources feature prevents certain resource types from being included in a region, but it does not prevent the resources from being placed inside the region unless you set the region’s Reserved property to On. To indicate to the Fitter that certain resources are not required inside a LogicLock region, define a resource filter. For more information about resource filters, refer to “LogicLock Resource Exclusions” in the Best Practices for Incremental Compilation Partitions and Floorplan Assignments chapter in volume 1 of the Quartus II Handbook.

Additional Quartus II LogicLock Design Features

To complement the LogicLock Regions window, the Quartus II software has additional features to help you design with LogicLock regions.

Analysis and Synthesis Resource Utilization by Entity

The Compilation Report contains an Analysis and Synthesis Resource Utilization by Entity section, which reports resource usage statistics, including entity-level information. You can use this feature to verify that any LogicLock region you manually create contains enough resources to accommodate all the entities you assign to it.
Quartus II Revisions Feature

When you evaluate different LogicLock regions in your design, you might want to experiment with different configurations to achieve your desired results. The Quartus II Revisions feature allows you to organize the same project with different settings until you find an optimum configuration.

To use the Revisions feature, on the Project menu, click Revisions. In the Revisions dialog box, you can create and specify revisions. You can create a revision from the current design or any previously created revisions. Each revision can have an associated description. You can use revisions to organize the placement constraints created for your LogicLock regions.

LogicLock Assignment Precedence

You can encounter conflicts during the assignment of entities and nodes to LogicLock regions. For example, an entire top-level entity might be assigned to one region and a node within this top-level entity assigned to another region. To resolve conflicting assignments, the Quartus II software maintains an order of precedence for LogicLock assignments. The following order of precedence, from highest to lowest, applies:

1. Exact node-level assignments
2. Path-based and wildcard assignments
3. Hierarchical assignments

For more information about LogicLock assignment precedence, refer to Understanding Assignment Priority in Quartus II Help.

Open the Priority dialog box by selecting Priority on the General tab of the LogicLock Regions Properties dialog box. You can change the priority of path-based and wildcard assignments with the Up and Down buttons in the Priority dialog box. To prioritize assignments between regions, you must select multiple LogicLock regions and then open the Priority dialog box from the LogicLock Regions Properties dialog box.

Virtual Pins

A virtual pin is an I/O element that is temporarily mapped to a logic element and not to a pin during compilation, and is then implemented as a LUT. Virtual pins should be used only for I/O elements in lower-level design entities that become nodes when imported to the top-level design. You can create virtual pins by assigning the Virtual Pin logic option to an I/O element.

You might use virtual pin assignments when you compile a partial design, because not all the I/Os from a partial design drive chip pins at the top level.

The virtual pin assignment identifies the I/O ports of a design module that are internal nodes in the top-level design. These assignments prevent the number of I/O ports in the lower-level modules from exceeding the total number of available device pins. Every I/O port that you designate as a virtual pin becomes mapped to either a logic cell or an adaptive logic module (ALM), depending on the target device.
The Virtual Pin logic option must be assigned to an input or output pin. If you assign this option to a bidirectional pin, tri-state pin, or registered I/O element, Analysis and Synthesis ignores the assignment. If you assign this option to a tri-state pin, the Fitter inserts an I/O buffer to account for the tri-state logic; therefore, the pin cannot be a virtual pin. You can use multiplexer logic instead of a tri-state pin if you want to continue to use the assigned pin as a virtual pin. Do not use tri-state logic except for signals that connect directly to device I/O pins.

In the top-level design, you connect these virtual pins to an internal node of another module. By making assignments to virtual pins, you can place those pins in the same location or region on the device as that of the corresponding internal nodes in the top-level module. You can use the Virtual Pin option when compiling a LogicLock module with more pins than the target device allows. The Virtual Pin option can enable timing analysis of a design module that more closely matches the performance of the module after you integrate it into the top-level design.

In the Node Finder, you can set Filter Type to Pins: Virtual to display all assigned virtual pins in the design. From the Assignment Editor, to access the Node Finder, double-click the To field; when the arrow appears on the right side of the field, click the arrow and select Node Finder.

Using LogicLock Regions in the Chip Planner

You can easily create LogicLock regions in the Chip Planner and assign resources to them.

Viewing Connections Between LogicLock Regions in the Chip Planner

You can view and edit LogicLock regions using the Chip Planner. To view and edit LogicLock regions, select the Floorplan Editing layer setting, or any layer setting that has the User-assigned LogicLock regions setting enabled.

The Chip Planner shows the connections between LogicLock regions. By default, you can view each connection as an individual line. You can choose to display connections between two LogicLock regions as a single bundled connection rather than as individual connection lines. To use this option, open the Chip Planner and on the View menu, click Inter-region Bundles.

For more information about the Inter-region Bundles dialog box, refer to Inter-region Bundles Dialog Box in Quartus II Help.

Using LogicLock Regions with the Design Partition Planner

You can optimize timing in a design by placing entities that share significant logical connectivity close to each other on the device. By default, the Fitter usually places closely connected entities in the same area of the device; however, you can use LogicLock regions, together with the Design Partition Planner and the Chip Planner, to help ensure that logically connected entities retain optimal placement from one compilation to the next.
You can view the logical connectivity between entities with the Design Partition Planner, and the physical placement of those entities with the Chip Planner. In the Design Partition Planner, you can identify entities that are highly interconnected, and place those entities in a partition. In the Chip Planner, you can create LogicLock regions and assign each partition to a LogicLock region, thereby preserving the placement of the entities.

For more information about using LogicLock regions with design partitions, refer to the *Quartus II Incremental Compilation for Hierarchical and Team-Based Design* and the *Best Practices for Incremental Compilation Partition and Floorplan Assignments* chapters in volume 1 of the *Quartus II Handbook*. For more information about using the Design Partition Planner with the Chip Planner, refer to *About the Design Partition Planner* and *Using the Design Partition Planner* in Quartus II Help.

**Design Floorplan Analysis Using the Chip Planner**

The Chip Planner helps you visually analyze the floorplan of your design at any stage of your design cycle. With the Chip Planner, you can view post-compilation placement, connections, and routing paths. You can also create LogicLock regions and location assignments. The Chip Planner allows you to create new logic cells and I/O atoms and to move existing logic cells and I/O atoms in your design. You can also see global and regional clock regions within the device, and the connections between I/O atoms, PLLs and the different clock regions.

From the Chip Planner, you can launch the Resource Property Editor, which you can use to change the properties and parameters of device resources, and modify connectivity between certain types of device resources. The Change Manager records any changes that you make to your design floorplan so that you can selectively undo changes if necessary.

For more information about the Resource Property Editor and the Change Manager, refer to the *Engineering Change Management with the Chip Planner* chapter in volume 2 of the *Quartus II Handbook*, and to *About the Resource Property Editor* and *About the Change Manager* in Quartus II Help.

The following sections present Chip Planner floorplan views and design analysis procedures which you can use with any predefined task, unless a procedure requires a specific task or editing mode.

**Chip Planner Floorplan Views**

The Chip Planner uses a hierarchical zoom viewer that shows various abstraction levels of the targeted Altera device. As you zoom in, the level of abstraction decreases, revealing more detail about your design.

For more information about Chip Planner floorplan views, refer to the *Engineering Change Management with the Chip Planner* chapter in volume 2 of the *Quartus II Handbook*.
**Bird’s Eye View**

The Bird’s Eye View displays a high-level picture of resource usage for the entire chip and provides a fast and efficient way to navigate between areas of interest in the Chip Planner.

The Bird’s Eye View is particularly useful when the parts of your design that you want to view are at opposite ends of the chip and you want to quickly navigate between resource elements without losing your frame of reference.

For more information about the Bird’s Eye View, refer to *Bird’s Eye View* and *Displaying Resources and Information* in Quartus II Help.

**Properties Window**

The Properties Window displays detailed properties of the objects (such as atoms, paths, LogicLock regions, or routing elements) currently selected in the Chip Planner. To display the Properties Window, click Properties on the View menu in the Chip Planner.

**Viewing Architecture-Specific Design Information**

By adjusting the Layer Settings in the Chip Planner, you can view the following architecture-specific information related to your design:

- **Device routing resources used by your design**—View how blocks are connected, as well as the signal routing that connects the blocks.
- **LE configuration**—View logic element (LE) configuration in your design. For example, you can view which LE inputs are used; if the LE utilizes the register, the look-up table (LUT), or both; as well as the signal flow through the LE.
- **ALM configuration**—View ALM configuration in your design. For example, you can view which ALM inputs are used, if the ALM utilizes the registers, the upper LUT, the lower LUT, or all of them. You can also view the signal flow through the ALM.
- **I/O configuration**—View device I/O resource usage. For example, you can view which components of the I/O resources are used, if the delay chain settings are enabled, which I/O standards are set, and the signal flow through the I/O.
- **PLL configuration**—View phase-locked loop (PLL) configuration in your design. For example, you can view which control signals of the PLL are used with the settings for your PLL.
- **Timing**—View the delay between the inputs and outputs of FPGA elements. For example, you can analyze the timing of the DATAB input to the COMBOUT output.

In addition, you can modify the following device properties with the Chip Planner:

- LEs and ALMs
- I/O cells
- PLLs
- Registers in RAM and DSP blocks
- Connections between elements
Placement of elements

For more information about LEs, ALMs, and other resources of an FPGA device, refer to the relevant device handbook.

Viewing Available Clock Networks in the Device

When you select a task with clock region layers enabled, you can display the areas of the chip that are driven by global and regional clock networks. This global clock display feature is available for Arria GX, Arria II, Cyclone II, Cyclone III, HardCopy II, HardCopy III, Stratix II, Stratix II GX, Stratix III, Stratix IV, and Stratix V device families.

Depending on the clock layers activated in the selected task, the Chip Planner displays regional and global clock regions in the device, and the connectivity between clock regions, pins, and PLLs. Clock regions appear as rectangular overlay boxes with labels indicating the clock type and index. You can select each clock network region by clicking on the clock region. The clock-shaped icon at the top-left corner indicates that the region represents a clock network region. You can change the color in which the Chip Planner displays clock regions on the Options dialog box of the Tools menu.

The Layer Settings dialog box lists layers for different clock region types; when the selected device does not contain a given clock region, the option for that category is unavailable in the dialog box. You can customize the Chip Planner’s display of clock regions by creating a custom task with selected clock layers enabled in the Layers Settings dialog box.

For more information about displaying clock regions, refer to Displaying Resources and Information in Quartus II Help.

Viewing Critical Paths

Critical paths are timing paths in your design that have a negative slack. These timing paths can span from device I/Os to internal registers, registers to registers, or from registers to device I/Os. The slack of a path determines its criticality; slack appears in the timing analysis report. Design analysis for timing closure is a fundamental requirement for optimal performance in highly complex designs. The analytical capability of the Chip Planner helps you close timing on complex designs.

Viewing critical paths in the Chip Planner helps you understand why a specific path is failing. You can see if any modification in the placement can reduce the negative slack. You can display details of a path (to expand/collapse the path to/from the connections in the path) by clicking Expand Connections in the toolbar, or by clicking on the “+/-” on the label.

You can locate failing paths from the timing report in the TimeQuest Timing Analyzer. To locate the critical paths, run the Report Timing task from the Custom Reports group in the Tasks pane of the TimeQuest Timing Analyzer. From the View pane, which lists the failing paths, right-click on any failing path or node, and select Locate Path. From the Locate dialog box, select Chip Planner to see the failing path in the Chip Planner.
To display paths in the floorplan, you must first make timing settings and perform a timing analysis.

For more information about performing static timing analysis with the Quartus II TimeQuest Timing Analyzer, refer to the *Quartus II TimeQuest Timing Analyzer* chapter in volume 3 of the *Quartus II Handbook*.

**Viewing Routing Congestion**

The **Routing Congestion** task allows you to determine the percentage of routing resources in use following a compilation. This feature can identify where there is a lack of routing resources, helping you to make design changes to meet routing congestion design requirements.

To view routing congestion in the Chip Planner, select the **Routing Congestion** task. The **Routing Utilization Settings** dialog box appears whenever you select the **Routing Congestion** task; this dialog box allows you to set a congestion threshold value, and to specify the types of routing interconnects of interest (Figure 15–3).

**Figure 15–3. Routing Utilization Settings dialog box**

For more information about displaying routing congestion, refer to *Displaying Resources and Information* in Quartus II Help.

The routing congestion map uses the color and shading of logic resources to indicate relative resource utilization; darker shading represents a greater utilization of routing resources (black indicates zero utilization). Areas where routing utilization exceeds the threshold value specified in the **Routing Utilization Settings** dialog box appear in red. The congestion map can help you determine whether you can modify the floorplan, or make changes to the RTL to reduce routing congestion.
The color and shading displayed by the congestion map for a particular area of the device is based on the total utilization of all interconnect types that you select in the Routing Utilization Settings dialog box. For example, consider the following routing utilization:

<table>
<thead>
<tr>
<th>Interconnect type</th>
<th>Total number of elements</th>
<th>Number of elements used</th>
<th>Percent utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>R3</td>
<td>216</td>
<td>69</td>
<td>32%</td>
</tr>
<tr>
<td>R6</td>
<td>108</td>
<td>71</td>
<td>66%</td>
</tr>
<tr>
<td>R24</td>
<td>48</td>
<td>46</td>
<td>96%</td>
</tr>
<tr>
<td>All interconnect</td>
<td>372</td>
<td>186</td>
<td>50%</td>
</tr>
</tbody>
</table>

If, in the Routing Utilization Settings dialog box, you select All interconnect, the color displayed in the congestion map corresponds to a utilization of 50%. If you select only R3 interconnect, the color displayed corresponds to 32%. If you select only R24, the color displayed corresponds to 96%.

To identify a lack of routing resources, it is necessary to investigate each routing interconnect type separately by selecting, in the Routing Utilization Settings dialog box, each interconnect type in turn.

**Viewing I/O Banks**

The Chip Planner can show all of the I/O banks of the device. To see the I/O bank map of the device, turn on the I/O Banks layer in the Layers Settings dialog box.

**Viewing High-Speed Serial Interfaces (HSSI)**

For the Stratix V device family, the Chip Planner displays a detailed block view of the receiver and transmitter channels of the high-speed serial interfaces. Figure 15–4 shows the blocks of a Stratix V HSSI receiver channel.

**Figure 15–4. Stratix V HSSI receiver channel**

**Generating Fan-In and Fan-Out Connections**

The ability to display fan-in and fan-out connections enables you to view the atoms that fan-in to or fan-out from the selected atom. To remove the connections displayed, use the Clear Unselected Connections icon in the Chip Planner toolbar.
Generating Immediate Fan-In and Fan-Out Connections

The ability to display immediate fan-in and fan-out connections enables you to view the resource that is the immediate fan-in or fan-out connection for the selected atom. For example, if you select a logic resource and choose to view the immediate fan-in for that resource, you can see the routing resource that drives the logic resource. You can generate immediate fan-in and fan-outs for all logic resources and routing resources. To remove the displayed connections from the screen, click the Clear Connections icon in the toolbar.

Highlight Routing

The Highlight Routing command enables you to highlight the routing resources used by a selected path or connection. Figure 15–5 shows the routing resources in use between two logic elements.

Figure 15–5. Highlight Routing

You can view and edit resources in the FPGA using the Resource Property Editor. For more information, refer to the Engineering Change Management with the Chip Planner chapter in volume 2 of the Quartus II Handbook.
Show Delays

With the Show Delays command, you can view timing delays for paths located from TimeQuest Timing Analyzer reports. For example, you can view the delay between two logic resources or between a logic resource and a routing resource. Figure 15–6 shows the delay associated with a path located from a TimeQuest Timing Analyzer report.

Figure 15–6. Show Delays

Exploring Paths in the Chip Planner

You can use the Chip Planner to explore paths between logic elements. The following example uses the Chip Planner to traverse paths from the Timing Analysis report.

Locate Path from the Timing Analysis Report to the Chip Planner

To locate a path from the Timing Analysis report to the Chip Planner, perform the following steps:

1. Select the path you want to locate.
2. Right-click the path in the Timing Analysis report, point to Locate, and click Locate in Chip Planner (Floorplan & Chip Editor). The path is displayed with its timing data in the Chip Planner main window and is listed in the Locate History window.

3. To view the routing resources taken for a path you have located in the Chip Planner, select the path and then click the Highlight Routing icon in the Chip Planner toolbar, or from the View menu, click Highlight Routing.

**Analyzing Connections for a Path**

To determine the connections between items in the Chip Planner, click the Expand Connections icon on the toolbar. To add the timing delays for paths located from the TimeQuest Timing Analyzer, click the Show Delays icon on the toolbar. Figure 15–7 shows the connections for a path located from the TimeQuest Timing Analyzer that are displayed in the Chip Planner. To see the constituent delays on the selected path, click on the “+” sign next to the path delay displayed in the Chip Planner.

**Figure 15–7. Path Analysis**

---

**Viewing Assignments in the Chip Planner**

You can view location assignments by selecting the appropriate layer set in the Chip Planner. To view location assignments, select the Floorplan Editing task or any custom task that displays block utilization, and the Assignment editing mode. See Figure 15–8.
The Chip Planner shows location assignments graphically, by displaying assigned resources in a particular color (gray, by default). You can create or move an assignment by dragging the selected resource to a new location.

Figure 15–8. Viewing Assignments in the Chip Planner

You can make node and pin location assignments and assignments to LogicLock regions and custom regions using the drag-and-drop method in the Chip Planner. The Fitter applies the assignments that you create during the next place-and-route operation.

For more information about managing assignments in the Chip Planner, refer to Working With Assignments in the Chip Planner in Quartus II Help.

**Viewing High-Speed and Low-Power Tiles in the Chip Planner**

The Chip Planner has a predefined task, Power, which shows the power map of Stratix III, Stratix IV, and Stratix V devices; these devices have ALMs that can operate in either high-speed mode or low-power mode. The power mode is set during the fitting process in the Quartus II software. These ALMs are grouped together to form larger blocks, called “tiles.”

To learn more about power analyses and optimizations in Stratix III devices, refer to AN 437: Power Optimization in Stratix III FPGAs. To learn more about power analyses and optimizations in Stratix IV devices, refer to AN 514: Power Optimization in Stratix IV FPGAs.
When you select the **Power** task in the Chip Planner for Stratix III, Stratix IV, or Stratix V devices, the Chip Planner displays low-power and high-speed tiles in contrasting colors; yellow tiles operate in a high-speed mode, while blue tiles operate in a low-power mode (see Figure 15–9). When you select the **Power** task, you can perform all floorplanner-related functions for this task; however, you cannot edit tiles to change the power mode.

**Figure 15–9. Viewing High-Speed and Low Power Tiles in a Stratix III Device**

---

**Scripting Support**

You can run procedures and specify the 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:

```bash
quartus_sh --qhelp
```

Information about scripting command options is also available in *API Functions for Tcl* in Quartus II Help.

For more information about Tcl scripting, refer to the *Tcl Scripting* chapter in volume 2 of the *Quartus II Handbook*. For more information about command-line scripting, refer to the *Command-Line Scripting* chapter in volume 2 of the *Quartus II Handbook*. For information about all settings and constraints in the Quartus II software, refer to the *Quartus II Settings File Manual*. 
Initializing and Uninitializing a LogicLock Region

You must initialize the LogicLock data structures before creating or modifying any LogicLock regions and before executing any of the Tcl commands listed below.

Use the following Tcl command to initialize the LogicLock data structures:

```tcl
initialize_logiclock
```

Use the following Tcl command to uninitialize the LogicLock data structures before closing your project:

```tcl
uninitialize_logiclock
```

Creating or Modifying LogicLock Regions

Use the following Tcl command to create or modify a LogicLock region:

```tcl
set_logiclock -auto_size true -floating true -region <my_region-name>
```

The command in the above example sets the size of the region to auto and the state to floating.

If you specify a region name that does not exist in the design, the command creates the region with the specified properties. If you specify the name of an existing region, the command changes all properties you specify and leaves unspecified properties unchanged.

For more information about creating LogicLock regions, refer to “Creating LogicLock Regions” on page 15–4.

Obtaining LogicLock Region Properties

Use the following Tcl command to obtain LogicLock region properties. This example returns the height of the region named `my_region`:

```tcl
get_logiclock -region my_region -height
```

Assigning LogicLock Region Content

Use the following Tcl commands to assign or change nodes and entities in a LogicLock region. This example assigns all nodes with names matching `fifo*` to the region named `my_region`.

```tcl
set_logiclock_contents -region my_region -to fifo*
```

You can also make path-based assignments with the following Tcl command:

```tcl
set_logiclock_contents -region my_region -from fifo -to ram*
```

Save a Node-Level Netlist for the Entire Design into a Persistent Source File

Make the following assignments to cause the Quartus II Fitter to save a node-level netlist for the entire design into a `.vqm` file:

```tcl
set_global_assignment -name LOGICLOCK_INCREMENTAL_COMPILE_ASSIGNMENT ON
set_global_assignment -name LOGICLOCK_INCREMENTAL_COMPILE_FILE <file name>
```
Any path specified in the file name is relative to the project directory. For example, specifying `atom_netlists/top.vqm` places `top.vqm` in the `atom_netlists` subdirectory of your project directory.

A `.vqm` file is saved in the directory specified at the completion of a full compilation.

The saving of a node-level netlist to a persistent source file is not supported for designs targeting newer devices such as Arria GX, Arria II, Cyclone III, MAX V, Stratix III, Stratix IV, or Stratix V.

### Setting LogicLock Assignment Priority

Use the following Tcl code to set the priority for a LogicLock region’s members. This example reverses the priorities of the LogicLock region in your design.

```tcl
set reverse [list]
for each member [get_logiclock_member_priority] {
    set reverse [insert $reverse 0 $member]
}
set_logiclock_member_priority $reverse
```

### Assigning Virtual Pins

Use the following Tcl command to turn on the virtual pin setting for a pin called `my_pin`:

```tcl
set_instance_assignment -name VIRTUAL_PIN ON -to my_pin
```

For more information about assigning virtual pins, refer to “Virtual Pins” on page 15–9.

For more information about Tcl scripting, refer to the `Tcl Scripting` chapter in volume 2 of the *Quartus II Handbook*.

### Conclusion

Design floorplan analysis is a valuable method for achieving timing closure and optimal performance in highly complex designs. With analysis capability, the Quartus II Chip Planner helps you close timing quickly on your designs. Using the Chip Planner together with LogicLock and Incremental Compilation enables you to compile your designs hierarchically, preserving the timing results from individual compilation runs. You can use LogicLock regions as part of an incremental compilation methodology to improve your productivity. You can also include a module in one or more projects while maintaining performance and reducing development costs and time to market. LogicLock region assignments give you complete control over logic and memory placement to improve the performance of nonhierarchical designs as well.
Document Revision History

Table 15–3 shows the revision history for this chapter.

Table 15–3. Document Revision History

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>May 2011</td>
<td>11.0.0</td>
<td>■ Updated for the 11.0 release.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Edited “LogicLock Regions”</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Updated “Viewing Routing Congestion”</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Updated “Locate History”</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Updated Figures 15-4, 15-9, 15-10, and 15-13</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Added Figure 15-6</td>
</tr>
<tr>
<td>December 2010</td>
<td>10.1.0</td>
<td>■ Updated for the 10.1 release.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Updated device support information</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Removed references to Timing Closure Floorplan; removed “Design Analysis Using the Timing Closure Floorplan” section</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Added links to online Help topics</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Added “Using LogicLock Regions with the Design Partition Planner” section</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Updated “Viewing Critical Paths” section</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Updated several graphics</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Updated format of Document revision History table</td>
</tr>
<tr>
<td>July 2010</td>
<td>10.0.0</td>
<td>■ Updated supported device information throughout</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Removed deprecated sections related to the Timing Closure Floorplan for older device families. (For information on using the Timing Closure Floorplan with older device families, refer to previous versions of the <em>Quartus II Handbook</em>, available in the <em>Quartus II Handbook Archive.</em>)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Updated “Creating Nonrectangular LogicLock Regions” section</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Added “Selected Elements Window” section</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Updated table 12-1</td>
</tr>
<tr>
<td>November 2009</td>
<td>9.1.0</td>
<td>■ Updated the following sections:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ “Chip Planner Tasks and Layers”</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ “LogicLock Regions”</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ “Back-Annotating LogicLock Regions”</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ “LogicLock Regions in the Timing Closure Floorplan”</td>
</tr>
<tr>
<td>May 2008</td>
<td>8.0.0</td>
<td>■ Added the following sections:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ “Reserve LogicLock Region”</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ “Creating Nonrectangular LogicLock Regions”</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ “Viewing Available Clock Networks in the Device”</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Updated Table 10–1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Removed the following sections:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Reserve LogicLock Region Design Analysis Using the Timing Closure Floorplan</td>
</tr>
</tbody>
</table>

For previous versions of the *Quartus II Handbook*, refer to the *Quartus II Handbook Archive*. 
Take an online survey to provide feedback about this handbook chapter.
16. Netlist Optimizations and Physical Synthesis

The Quartus® II software offers physical synthesis optimizations to improve your design beyond the optimization performed in the normal course of the Quartus II compilation flow.

Physical synthesis optimizations can help improve the performance of your design regardless of the synthesis tool used, although the effect of physical synthesis optimizations depends on the structure of your design.

Netlist optimization options work with the atom netlist of your design, which describes a design in terms of Altera®-specific primitives. An atom netlist file can be an Electronic Design Interchange Format (.edf) file or a Verilog Quartus Mapping (.vqm) file generated by a third-party synthesis tool, or a netlist used internally by the Quartus II software. Physical synthesis optimizations are applied at different stages of the Quartus II compilation flow, either during synthesis, fitting, or both.

This chapter explains how the physical synthesis optimizations in the Quartus II software can modify your design’s netlist to improve the quality of results. This chapter also provides information about preserving compilation results through back-annotation and writing out a new netlist, and provides guidelines for applying the various options.

Because the node names for primitives in the design can change when you use physical synthesis optimizations, you should evaluate whether your design flow requires fixed node names. If you use a verification flow that might require fixed node names, such as the SignalTap® II Logic Analyzer, formal verification, or the LogicLock based optimization flow (for legacy devices), you must turn off physical synthesis options.

WYSIWYG Primitive Resynthesis

If you use a third-party tool to synthesize your design, use the Perform WYSIWYG primitive resynthesis option to apply optimizations to the synthesized netlist.

The Perform WYSIWYG primitive resynthesis option directs the Quartus II software to un-map the logic elements (LEs) in an atom netlist to logic gates, and then re-map the gates back to Altera-specific primitives. Third-party synthesis tools generate either an .edf or .vqm atom netlist file using Altera-specific primitives. When you turn on the Perform WYSIWYG primitive resynthesis option, the Quartus II software can work on different techniques specific to the device architecture during the re-mapping process. This feature re-maps the design using the Optimization Technique specified for your project (Speed, Area, or Balanced).

The Perform WYSIWYG primitive resynthesis option has no effect if you are using Quartus II integrated synthesis to synthesize your design.
To turn on the **Perform WYSIWYG primitive resynthesis** option, perform the following steps:

1. On the Assignments menu, click **Settings**. The **Settings** dialog box appears.
2. In the **Category** list, select **Analysis and Synthesis Settings**. The **Analysis & Synthesis Settings** page appears.
3. Turn on **Perform WYSIWYG Primitive Resynthesis**, and click **OK**.

If you want to perform WYSIWYG resynthesis on only a portion of your design, you can use the Assignment Editor to assign the **Perform WYSIWYG primitive resynthesis** logic option to a lower-level entity in your design. This logic option is available for all Altera devices supported by the Quartus II software except MAX 3000 and MAX 7000 devices.

The results of the remapping depend on the **Optimization Technique** you choose. To select an **Optimization Technique**, perform the following steps:

1. In the **Category** list, select **Analysis & Synthesis Settings**. The **Analysis & Synthesis Settings** page appears.
2. Under **Optimization Technique**, select **Speed**, **Area**, or **Balanced** to specify how the Quartus II technology mapper optimizes the design. The **Balanced** setting is the default for many Altera device families; this setting optimizes the timing critical parts of the design for speed and the rest of the design for area.
3. Click **OK**.

Refer to the **Quartus II Integrated Synthesis** chapter in volume 1 of the **Quartus II Handbook** for details on the Optimization Technique option.

**Figure 16–1** shows the Quartus II software flow for the WYSIWYG primitive resynthesis feature.
The Perform WYSIWYG primitive resynthesis option unmaps and remaps only logic cells, also referred to as LCELL or LE primitives, and regular I/O primitives (which may contain registers). Double data rate (DDR) I/O primitives, memory primitives, digital signal processing (DSP) primitives, and logic cells in carry/cascade chains are not remapped. Logic specified in an encrypted .vqm file or an .edf file, such as third-party intellectual property (IP), is not touched.

The Perform WYSIWYG primitive resynthesis option can change node names in the .vqm file or .edf file from your third-party synthesis tool, because the primitives in the atom netlist are broken apart and then remapped by the Quartus II software. The remapping process removes duplicate registers, but registers that are not removed retain the same name after remapping.

Any nodes or entities that have the Netlist Optimizations logic option set to Never Allow are not affected during WYSIWYG primitive resynthesis. You can use the Assignment Editor to apply the Netlist Optimizations logic option. This option disables WYSIWYG resynthesis for parts of your design.

Primitive node names are specified during synthesis. When netlist optimizations are applied, node names might change because primitives are created and removed. HDL attributes applied to preserve logic in third-party synthesis tools cannot be maintained because those attributes are not written into the atom netlist read by the Quartus II software.

If you use the Quartus II software to synthesize, you can use the Preserve Register (preserve) and Keep Combinational Logic (keep) attributes to maintain certain nodes in the design.

For more information about using these attributes during synthesis in the Quartus II software, refer to the Quartus II Integrated Synthesis chapter in volume 1 of the Quartus II Handbook.

### Performing Physical Synthesis Optimizations

The Quartus II design flow involves separate steps of synthesis and fitting. The synthesis step optimizes the logical structure of a circuit for area, speed, or both. The Fitter then places and routes the logic cells to ensure critical portions of logic are close together and use the fastest possible routing resources. While you are using this push-button flow, the synthesis stage is unable to anticipate the routing delays seen in the Fitter. Because routing delays are a significant part of the typical critical path delay, the physical synthesis optimizations available in the Quartus II software take those routing delays into consideration and focus timing-driven optimizations at those parts of the design. This tight integration of the fitting and synthesis processes is known as physical synthesis.

The following sections describe the physical synthesis optimizations available in the Quartus II software, and how they can help improve your performance results. Physical synthesis optimization options can be used with Arria series, Cyclone, HardCopy, and Stratix series device families.
If you are migrating your design to a HardCopy II device, you can target physical synthesis optimizations to the FPGA architecture in the FPGA-first flow or to the HardCopy II architecture in the HardCopy-first flow. The optimizations are mapped to the other device architecture during the migration process.

You cannot target optimizations to both device architectures individually because doing so results in a different post-fitting netlist for each device.

For more information about physical synthesis optimizations, refer to *Physical Synthesis Optimizations Page (Settings Dialog Box)* in Quartus II Help. For more information about using physical synthesis with HardCopy devices, refer to the *Quartus II Support for HardCopy Series Devices* chapter in volume 1 of the *Quartus II Handbook*.

You can choose the physical synthesis optimization options you want for your design during synthesis and fitting in the *Physical Synthesis Optimizations* page under the *Compilation Process Settings* page in the *Settings* dialog box. The settings include optimizations for improving performance and fitting in the selected device.

You can also set the effort level for physical synthesis optimizations. Normally, physical synthesis optimizations increase the compilation time; however, you can select the *Fast* effort level if you want to limit the increase in compilation time. When you select the *Fast* effort level, the Quartus II software performs limited register retiming operations during fitting. The *Extra* effort level runs additional algorithms to get the best circuit performance, but results in increased compilation time.

To optimize performance, the following options are available:

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

To optimize for better fitting, you can choose from the following options:

- Perform physical synthesis for combinational logic
- Perform logic to memory mapping

To view and modify the physical synthesis optimization options, perform the following steps:

1. On the Assignments menu, click *Settings*. The *Settings* dialog box appears.
2. In the *Category* list, select *Physical Synthesis Optimizations* under *Compilation Process Settings*. The *Physical Synthesis Optimizations* page appears.
3. Specify the options for performing physical synthesis optimizations.

Some physical synthesis options affect only registered logic and some options affect only combinational logic. Select options based on whether you want to keep the registers intact or not. For example, if your verification flow involves formal verification, you might have to keep the registers intact.
Performing Physical Synthesis Optimizations

All Physical Synthesis optimizations write results to the **Netlist Optimizations** report, which provides a list of atom netlist files that were modified, created, and deleted during physical synthesis. To access the **Netlist Optimizations** report, perform the following steps:

1. On the Processing menu, click **Compilation Report**.
2. In the **Compilation Report** list, select **Netlist Optimizations** under **Fitter**.

Similarly, physical synthesis optimizations performed during synthesis write results to the synthesis report. To access this report, perform the following steps:

1. On the Processing menu, click **Compilation Report**.
2. In the **Compilation Report** list, select **Analysis & Synthesis**.
3. In the **Optimization Results** folder, select **Netlist Optimizations**. The **Physical Synthesis Netlist Optimizations** table appears, listing the physical synthesis netlist optimizations performed during synthesis.

Nodes or entities that have the **Netlist Optimizations** logic option set to **Never Allow** are not affected by the physical synthesis algorithms. You can use the Assignment Editor to apply the **Netlist Optimizations** logic option. Use this option to disable physical synthesis optimizations for parts of your design.

**Automatic Asynchronous Signal Pipelining**

The **Perform automatic asynchronous signal pipelining** option on the **Physical Synthesis Optimizations** page in the **Compilation Process Settings** section of the **Settings** dialog box allows the Quartus II Fitter to perform automatic insertion of pipeline stages for asynchronous clear and asynchronous load signals during fitting when these signals negatively affect performance. You can use this option if asynchronous control signal recovery and removal times are not achieving their requirements.

The **Perform automatic asynchronous signal pipelining** option improves performance for designs in which asynchronous signals in very fast clock domains cannot be distributed across the chip fast enough due to long global network delays. This optimization performs automatic pipelining of these signals, while attempting to minimize the total number of registers inserted.

The **Perform automatic asynchronous signal pipelining** option adds registers to nets driving the asynchronous clear or asynchronous load ports of registers. These additional registers add register delays (adds latency) to the reset, adding the same number of register delays for each destination using the reset. The additional register delays can change the behavior of the signal in the design; therefore, you should use this option only if additional latency on the reset signals does not violate any design requirements. This option also prevents the promotion of signals to global routing resources.

The Quartus II software performs automatic asynchronous signal pipelining only if **Enable Recovery/Removal analysis** is turned on. If you use the TimeQuest Timing Analyzer, **Enable Recovery/Removal analysis** is turned on by default. Pipelining is allowed only on asynchronous signals that have the following properties:

- The asynchronous signal is synchronized to a clock (a synchronization register drives the signal)
The asynchronous signal fans-out only to asynchronous control ports of registers. The Quartus II software does not perform automatic asynchronous signal pipelining on asynchronous signals that have the Netlist Optimization logic option set to Never Allow.

**Physical Synthesis for Combinational Logic**

To optimize the design and reduce delay along critical paths, you can turn on the Perform physical synthesis for combinational logic option, which swaps the look-up table (LUT) ports within LEs so that the critical path has fewer layers through which to travel. The Perform physical synthesis for combinational logic option also allows the duplication of LUTs to enable further optimizations on the critical path.

For more information about using the Perform physical synthesis for combinational logic option, refer to Physical Synthesis Optimizations Page (Settings Dialog Box) and to Setting Up and Running the Fitter in Quartus II Help.

The Perform physical synthesis for combinational logic option affects only combinational logic in the form of LUTs. These transformations might occur during the synthesis stage or the Fitter stage during compilation. The registers contained in the affected logic cells are not modified. Inputs into memory blocks, DSP blocks, and I/O elements (IOEs) are not swapped.

The Quartus II software does not perform combinational optimization on logic cells that have the following properties:

- Are part of a chain
- Drive global signals
- Are constrained to a single logic array block (LAB) location
- Have the Netlist Optimizations option set to Never Allow

If you want to consider logic cells with any of these conditions for physical synthesis, you can override these rules by setting the Netlist Optimizations logic option to Always Allow on a given set of nodes.

**Physical Synthesis for Registers—Register Duplication**

The Perform register duplication option on the Physical Synthesis Optimizations page in the Compilation Process Settings section of the Settings dialog box allows the Quartus II Fitter to duplicate registers based on Fitter placement information. You can also duplicate combinational logic when this option is enabled. A logic cell that fans out to multiple locations can be duplicated to reduce the delay of one path without degrading the delay of another. The new logic cell can be placed closer to critical logic without affecting the other fan-out paths of the original logic cell.

For more information about the Perform register duplication option, refer to Physical Synthesis Optimizations Page (Settings Dialog Box) and to Setting Up and Running the Fitter in Quartus II Help.
The Quartus II software does not perform register duplication on logic cells that have the following properties:

- Are part of a chain
- Contain registers that drive asynchronous control signals on another register
- Contain registers that drive the clock of another register
- Contain registers that drive global signals
- Contain registers that are constrained to a single LAB location
- Contain registers that are driven by input pins without a \( t_{SU} \) constraint
- Contain registers that are driven by a register in another clock domain
- Are considered virtual I/O pins
- Have the Netlist Optimizations option set to Never Allow

For more information about virtual I/O pins, refer to the Analyzing and Optimizing the Design Floorplan chapter in volume 2 of the Quartus II Handbook.

If you want to consider logic cells that meet any of these conditions for physical synthesis, you can override these rules by setting the Netlist Optimizations logic option to Always Allow on a given set of nodes.

**Physical Synthesis for Registers—Register Retiming**

The Perform Register Retiming option enables the movement of registers across combinational logic, allowing the Quartus II software to trade off the delay between timing-critical paths and non-critical paths. Register retiming can be done during Quartus II integrated synthesis or during the Fitter stages of design compilation.

Figure 16–2 shows an example of register retiming in which the 10-ns critical delay is reduced by moving the register relative to the combinational logic.

**Figure 16–2. Register Retiming Diagram**

Retiming can create multiple registers at the input of a combinational block from a register at the output of a combinational block. In this case, the new registers have the same clock and clock enable. The asynchronous control signals and power-up level are derived from previous registers to provide equivalent functionality. Retiming can also combine multiple registers at the input of a combinational block to a single register (Figure 16–3).
Performing Physical Synthesis Optimizations

To move registers across combinational logic to balance timing, perform the following steps:

1. On the Assignments menu, click Settings. The Settings dialog box appears.


3. Specify your preferred option under Optimize for performance (physical synthesis) and Effort level.

4. Click OK.

For more information about the Optimize for performance (physical synthesis) options and effort levels, refer to Physical Synthesis Optimizations Page (Settings Dialog Box) in Quartus II Help.

If you want to prevent register movement during register retiming, you can set the Netlist Optimizations logic option to Never Allow. You can apply this option to either individual registers or entities in the design using the Assignment Editor.

In digital circuits, synchronization registers are instantiated on cross clock domain paths to reduce the possibility of metastability. The Quartus II software detects such synchronization registers and does not move them, even if register retiming is turned on.

The following sets of registers are not moved during register retiming:

- Both registers in a direct connection from input pin-to-register-to-register if both registers have the same clock and the first register does not fan-out to anywhere else. These registers are considered synchronization registers.

- Both registers in a direct connection from register-to-register if both registers have the same clock, the first register does not fan out to anywhere else, and the first register is fed by another register in a different clock domain (directly or through combinational logic). These registers are considered synchronization registers.

The Quartus II software assumes that a synchronization register chain consists of two registers. If your design has synchronization register chains with more than two registers, you must indicate the number of registers in your synchronization chains so that they are not affected by register retiming. To do this, perform the following steps:

1. On the Assignments menu, click Settings. The Settings dialog box appears.

2. In the Category list, select Analysis & Synthesis Settings. The Analysis & Synthesis Setting page appears.

Figure 16–3. Combining Registers with Register Retiming

4. In the Name list, select Synchronization Register Chain Length and modify the setting to match the synchronization register length used in your design. If you set a value of 1 for the Synchronization Register Chain Length, it means that any registers connected to the first register in a register-to-register connection can be moved during retiming. A value of \( n > 1 \) means that any registers in a sequence of length 1, 2, \( \ldots \) \( n \) are not moved during register retiming.

The Quartus II software does not perform register retiming on logic cells that have the following properties:

- Are part of a cascade chain
- Contain registers that drive asynchronous control signals on another register
- Contain registers that drive the clock of another register
- Contain registers that drive a register in another clock domain
- Contain registers that are driven by a register in another clock domain

The Quartus II software does not usually retime registers across different clock domains; however, if you use the Classic Timing Analyzer and specify a global \( f_{\text{MAX}} \) requirement, the Quartus II software interprets all clocks as related. Consequently, the Quartus II software might try to retime register-to-register paths associated with different clocks.

To avoid this circumstance, provide individual \( f_{\text{MAX}} \) requirements to each clock when using Classic Timing Analysis. When you constrain each clock individually, the Quartus II software assumes no relationship between different clock domains and considers each clock domain to be asynchronous to other clock domains; hence no register-to-register paths crossing clock domains are retimed.

When you use the TimeQuest Timing Analyzer, register-to-register paths across clock domains are never retimed, because the TimeQuest Timing Analyzer treats all clock domains as asynchronous to each other unless they are intentionally grouped.

- Contain registers that are constrained to a single LAB location
- Contain registers that are connected to SERDES
- Are considered virtual I/O pins
- Registers that have the Netlist Optimizations logic option set to Never Allow

For more information about virtual I/O pins, refer to the Analyzing and Optimizing the Design Floorplan chapter in volume 2 of the Quartus II Handbook.

If you want to consider logic cells that meet any of these conditions for physical synthesis, you can override these rules by setting the Netlist Optimizations logic option to Always Allow on a given set of registers.
Preserving Your Physical Synthesis Results

The Quartus II software generates the same results on every compilation for the same source code and settings on a given system, hence you do not need to preserve your results from compilation to compilation. When you make changes to the source code or to the settings, you usually get the best results by allowing the software to compile without using previous compilation results or location assignments. In some cases, if you avoid performing analysis and synthesis or `quartus_map`, and run the Fitter or another desired Quartus II executable instead, you can skip the synthesis stage of the compilation.

When you use the Quartus II incremental compilation flow, you can preserve synthesis results for a particular partition of your design by choosing a netlist type of post-synthesis. If you want to preserve fitting results between compilation runs, choose a netlist type of post-fit during incremental compilation.

The rest of this section is relevant only for those designs using older devices that do not support incremental compilation.

For information about the incremental compilation design methodology, refer to the `Quartus II Incremental Compilation for Hierarchical and Team-Based Design` chapter in volume 1 of the `Quartus II Handbook`, and to `About Incremental Compilation` in Quartus II Help.

You can preserve the resulting nodes from physical synthesis in older devices that do not support incremental compilation. You might need to preserve nodes if you use the LogicLock flow to back-annotate placement, import one design into another, or both. For all device families that support incremental compilation, use that feature to preserve results.

To preserve the nodes from Quartus II physical synthesis optimization options for older devices that do not support incremental compilation (such as Max II devices), perform the following steps:

1. On the Assignments menu, click Settings. The Settings dialog box appears.
3. Turn on Save a node-level netlist of the entire design into a persistent source file. This setting is not available for Cyclone III, Stratix III, and newer devices.
4. Click OK.

The Save a node-level netlist of the entire design into a persistent source file option saves your final results as an atom-based netlist in .vqm file format. By default, the Quartus II software places the .vqm file in the atom_netlists directory under the current project directory. To create a different .vqm file using different Quartus II settings, in the Compilation Process Settings page, change the File name setting.
Performing Physical Synthesis Optimizations

If you use the physical synthesis optimizations and want to lock down the location of all LEs and other device resources in the design with the **Back-Annotate Assignments** command, a `.vqm` file netlist is required. The `.vqm` file preserves the changes that you made to your original netlist. Because the physical synthesis optimizations depend on the placement of the nodes in the design, back-annotating the placement changes the results from physical synthesis. Changing the results means that node names are different, and your back-annotated locations are no longer valid.

You should not use a Quartus II-generated `.vqm` file or back-annotated location assignments with physical synthesis optimizations unless you have finalized the design. Making any changes to the design invalidates your physical synthesis results and back-annotated location assignments. If you require changes later, use the new source HDL code as your input files, and remove the back-annotated assignments corresponding to the Quartus II-generated `.vqm` file.

To back-annotate logic locations for a design that was compiled with physical synthesis optimizations, first create a `.vqm` file. When recompiling the design with the hard logic location assignments, use the new `.vqm` file as the input source file and turn off the physical synthesis optimizations for the new compilation.

If you are importing a `.vqm` file and back-annotated locations into another project that has any **Netlist Optimizations** turned on, you must apply the **Never Allow** constraint to make sure node names don’t change; otherwise, the back-annotated location or LogicLock assignments are invalid.

For newer devices, such as the Arria, Cyclone, or Stratix series, use incremental compilation to preserve compilation results instead of using logic back-annotation.

### Physical Synthesis Options for Fitting

The Quartus II software provides physical synthesis optimization options for improving fitting results. To access these options, perform the following steps:

1. On the Assignments menu, click **Settings**. The **Settings** dialog box appears.
2. In the **Category** list, select **Physical Synthesis Optimizations under Compilation Process Settings**. The **Physical Synthesis Optimizations** page appears.
3. Under **Optimize for fitting** (physical synthesis for density), there are two physical synthesis options available to improve fitting your design in the target device:
   - **Physical synthesis for combinational logic** and **Perform logic to memory mapping** (Table 16–1).

#### Table 16–1. Physical Synthesis Optimizations Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physical Synthesis for Combinational Logic</td>
<td>When you select this option, the Fitter detects duplicate combinational logic and optimizes combinational logic to improve the fit.</td>
</tr>
<tr>
<td>Perform Logic to Memory Mapping</td>
<td>When you select this option, the Fitter can remap registers and combinational logic in your design into unused memory blocks and achieves a fit.</td>
</tr>
</tbody>
</table>

For more information about physical synthesis optimization options, refer to **Physical Synthesis Optimizations Page (Settings Dialog Box)** in Quartus II Help.
Applying Netlist Optimization Options

The improvement in performance when using netlist optimizations is design dependent. If you have restructured your design to balance critical path delays, netlist optimizations might yield minimal improvement in performance. You may have to experiment with available options to see which combination of settings works best for a particular design. Refer to the messages in the compilation report to see the magnitude of improvement with each option, and to help you decide whether you should turn on a given option or specific effort level.

Turning on more netlist optimization options can result in more changes to the node names in the design; bear this in mind if you are using a verification flow, such as the SignalTap II Logic Analyzer or formal verification that requires fixed or known node names.

Applying all of the physical synthesis options at the Extra effort level generally produces the best results for those options, but adds significantly to the compilation time. You can also use the Physical synthesis effort level options to decrease the compilation time. The WYSIWYG primitive resynthesis option does not add much compilation time relative to the overall design compilation time.

To find the best results, you can use the Quartus II Design Space Explorer (DSE) to apply various sets of netlist optimization options.

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

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 in volume 2 of the Quartus II Handbook and API Functions for Tcl in Quartus II Help. Refer to the Quartus II Settings File Manual for information about all settings and constraints in the Quartus II software. For more information about command-line scripting, refer to the Command-Line Scripting chapter in volume 2 of the Quartus II Handbook.

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

Use the following Tcl command to make a global assignment:

```
set_global_assignment -name <QSF variable name> <value>
```

Use the following Tcl command to make an instance assignment:

```
set_instance_assignment -name <QSF variable name> <value> -to <instance name>
```
Synthesis Netlist Optimizations

Table 16–2 lists the Quartus II Settings File (.qsf) variable names and applicable values for the settings discussed in “WYSIWYG Primitive Resynthesis” on page 16–1. The .qsf file variable name is used 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>
<thead>
<tr>
<th>Setting Name</th>
<th>Quartus II Settings 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>Optimization Technique</td>
<td>&lt;Device Family Name&gt;_OPTIMIZATION_TECHNIQUE</td>
<td>AREA, SPEED, BALANCED</td>
<td>Global, Instance</td>
</tr>
<tr>
<td>Power-Up Don't Care</td>
<td>ALLOW_POWER_UP_DONT_CARE</td>
<td>ON, OFF</td>
<td>Global</td>
</tr>
<tr>
<td>Save a node-level netlist into a persistent source file</td>
<td>LOGICLOCK_INCREMENTAL_COMPILE_ASSIGNMENT</td>
<td>ON, OFF</td>
<td>Global</td>
</tr>
<tr>
<td>Allow Netlist Optimizations</td>
<td>ADV_NETLIST_OPT_ALLOWED</td>
<td>&quot;ALWAYS ALLOW&quot;, &quot;DEFAULT,&quot; &quot;NEVER ALLOW&quot;</td>
<td>Instance</td>
</tr>
</tbody>
</table>

Physical Synthesis Optimizations

Table 16–3 lists the .qsf file variable name and applicable values for the settings discussed in “Performing Physical Synthesis Optimizations” on page 16–3. The .qsf file variable name is used 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>
<thead>
<tr>
<th>Setting Name</th>
<th>Quartus II Settings File Variable Name</th>
<th>Values</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physical Synthesis for Combinational Logic</td>
<td>PHYSICAL_SYNTHESIS_COMBO_LOGIC</td>
<td>ON, OFF</td>
<td>Global</td>
</tr>
<tr>
<td>Automatic Asynchronous Signal Pipelining</td>
<td>PHYSICAL_SYNTHESISASYNCHRONOUS_SIGNAL_PIPELINING</td>
<td>ON, OFF</td>
<td>Global</td>
</tr>
<tr>
<td>Perform Register Duplication</td>
<td>PHYSICAL_SYNTHESIS_REGISTER_DUPLICATION</td>
<td>ON, OFF</td>
<td>Global</td>
</tr>
<tr>
<td>Perform Register Retiming</td>
<td>PHYSICAL_SYNTHESIS_REGISTER_RETIMING</td>
<td>ON, OFF</td>
<td>Global</td>
</tr>
<tr>
<td>Power-Up Don’t Care</td>
<td>ALLOW_POWER_UP_DONT_CARE</td>
<td>ON, OFF</td>
<td>Global, Instance</td>
</tr>
<tr>
<td>Power-Up Level</td>
<td>POWER_UP_LEVEL</td>
<td>HIGH, LOW</td>
<td>Instance</td>
</tr>
</tbody>
</table>
Conclusion

Physical synthesis optimizations restructure and optimize your design netlist. You can take advantage of these Quartus II netlist optimizations to help improve your quality of results.

Incremental Compilation

For information about scripting and command line usage for incremental compilation as mentioned in “Preserving Your Physical Synthesis Results” on page 16–10, refer to the Quartus II Incremental Compilation for Hierarchical and Team-Based Design chapter in volume 1 of the Quartus II Handbook.

Back-Annotating Assignments

You can use the `logiclock_back_annotate` Tcl command to back-annotate resources in your design. This command can back-annotate resources in LogicLock regions, and resources in designs without LogicLock regions.

For more information about back-annotating assignments, refer to “Preserving Your Physical Synthesis Results” on page 16–10.

The following Tcl command back-annotates all registers in your design:

```
logiclock_back_annotate -resource_filter "REGISTER"
```

The `logiclock_back_annotate` command is in the `backannotate` package.

Conclusion

Table 16–4 shows the revision history for this chapter.

## Document Revision History

Table 16–4. Document Revision History (Part 1 of 2)

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>December 2010</td>
<td>10.0.1</td>
<td>Template update.</td>
</tr>
<tr>
<td>July 2010</td>
<td>10.0.0</td>
<td>Added links to Quartus II Help in several sections.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Removed Referenced Documents section.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reformatted Document Revision History</td>
</tr>
</tbody>
</table>
Table 16–4. Document Revision History (Part 2 of 2)

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Changes</th>
</tr>
</thead>
</table>
| November 2009 | 9.1.0   | • Added information to “Physical Synthesis for Registers—Register Retiming”  
            |         | • Added information to “Applying Netlist Optimization Options”            
            |         | • Made minor editorial updates                                          |
| March 2009  | 9.0.0   | • Was chapter 11 in the 8.1.0 release.                                 
            |         | • Updated the “Physical Synthesis for Registers—Register Retiming”       
            |         | • “Physical Synthesis Options for Fitting”                              
            |         | • Updated “Performing Physical Synthesis Optimizations”                  
            |         | • Deleted Gate-Level Register Retiming section.                         
            |         | • Updated the referenced documents                                     |
| November 2008 | 8.1.0   | • Changed to 8½” × 11” page size. No change to content.                 |
| May 2008    | 8.0.0   | • Updated “Physical Synthesis Optimizations for Performance on page 11-9  
            |         | • Added Physical Synthesis Options for Fitting on page 11-16             |

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

Take an online survey to provide feedback about this handbook chapter.