

This chapter provides information and design scenarios to help you partition your design to take advantage of the Quartus® II incremental compilation feature.

The ability to iterate rapidly through FPGA design and debugging stages is critical. The Quartus II software introduced the FPGA industry's first true incremental design and compilation flow, with the following benefits:

- Preserves the results and performance for unchanged logic in your design as you make changes elsewhere.
- Reduces design iteration time by an average of 75% for small changes in large designs, so that you can perform more design iterations per day and achieve timing closure efficiently.
- Facilitates modular hierarchical and team-based design flows, as well as design reuse and intellectual property (IP) delivery.



Quartus II incremental compilation supports the Arria®, Stratix®, and Cyclone® series of devices.

This document contains the following sections:

- “Deciding Whether to Use an Incremental Compilation Flow” on page 3–1
- “Incremental Compilation Summary” on page 3–7
- “Common Design Scenarios Using Incremental Compilation” on page 3–10
- “Deciding Which Design Blocks Should Be Design Partitions” on page 3–19
- “Specifying the Level of Results Preservation for Subsequent Compilations” on page 3–25
- “Exporting Design Partitions from Separate Quartus II Projects” on page 3–30
- “Team-Based Design Optimization and Third-Party IP Delivery Scenarios” on page 3–39
- “Creating a Design Floorplan With LogicLock Regions” on page 3–48
- “Incremental Compilation Restrictions” on page 3–51
- “Scripting Support” on page 3–57

Deciding Whether to Use an Incremental Compilation Flow

The Quartus II incremental compilation feature enhances the standard Quartus II design flow by allowing you to preserve satisfactory compilation results and performance of unchanged blocks of your design.

© 2013 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, HARDCOPY, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at www.altera.com/common/legal.html. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.



This section outlines the flat compilation flow with no design partitions, the incremental flow when you divide the design into partitions, and the differences between the flat compilation and incremental compilation flows. This section also explains when a flat compilation flow is satisfactory, and highlights some of the reasons why you might want to create design partitions and use the incremental compilation flow. A discussion about incremental and team design flows in “[Team-Based Design Flows and IP Delivery](#)” on page 3–6 describes how it is beneficial to keep your design within one project, as well as when it might be necessary for other team members or IP providers to develop particular design blocks or partitions separately, and then later integrate their partitions into the top-level design.

Flat Compilation Flow with No Design Partitions

In the flat compilation flow with no design partitions, all the source code is processed and mapped during the Analysis and Synthesis stage, and placed and routed during the Fitter stage whenever the design is recompiled after a change in any part of the design. One reason for this behavior is to ensure optimal push-button quality of results. By processing the entire design, the Compiler can perform global optimizations to improve area and performance.

You can use a flat compilation flow for small designs, such as designs in CPLD devices or low-density FPGA devices, when the timing requirements are met easily with a single compilation. A flat design is satisfactory when compilation time and preserving results for timing closure are not concerns.



For more information on how to reduce compilation time when you use a flat compilation for your design, refer to the [Reducing Compilation Time](#) chapter in volume 2 of the *Quartus II Handbook*.

Incremental Capabilities Available When A Design Has No Partitions

The Quartus II software has incremental compilation features available even when you do not partition your design, including Smart Compilation, incremental debugging, and Rapid Recompile. These features work in either an incremental or flat compilation flow.

In any Quartus II compilation flow, you can use Smart Compilation to allow the Compiler to determine which compilation stages are required, based on the changes made to the design since the last smart compilation, and then skip any stages that are not required. For example, when Smart Compilation is turned on, the Compiler skips the Analysis and Synthesis stage if all the design source files are unchanged. When Smart Compilation is turned on, if you make any changes to the logic of a design, the Compiler does not skip any compilation stage. You can turn on Smart Compilation on the **Compilation Process Settings** page of the **Setting** dialog box.

The Quartus II software also includes a Rapid Recompile feature that instructs the Compiler to reuse the compatible compilation results if most of the design has not changed since the last compilation. This feature reduces compilation times for small and isolated design changes. You do not have control over which parts of the design are recompiled using this option; the Compiler determines which parts of the design must be recompiled. The Rapid Recompile feature preserves performance and can save compilation time by reducing the amount of changed logic that must be recompiled.

- ❓ For more information on Rapid Recompile, refer to *About Rapid Recompile* in Quartus II Help.

During the debugging stage of the design cycle, you can use incremental compilation to add the SignalTap® II Logic Analyzer incrementally to your design, even if the design does not have partitions. To preserve the compilation netlist for the entire design, instruct the software to reuse the compilation results for the automatically-created "Top" partition that contains the entire design. For more information, refer to "Debugging Incrementally With the SignalTap II Logic Analyzer" on page 3-13.

Incremental Compilation Flow With Design Partitions

In the standard incremental compilation design flow, the top-level design is divided into design partitions, which can be compiled and optimized together in the top-level Quartus II project. You can preserve fitting results and performance for completed partitions while other parts of the design are changing, which reduces the compilation times for each design iteration.

Incremental compilation is recommended for large designs and high resource densities when preserving results is important to achieve timing closure. The incremental compilation feature also facilitates team-based design flows that allow designers to create and optimize design blocks independently, when necessary. Refer to "Team-Based Design Flows and IP Delivery" on page 3-6 for more information.

To take advantage of incremental compilation, start by splitting your design along any of its hierarchical boundaries into design blocks to be compiled incrementally, and set each block as a design partition. The Quartus II software synthesizes each individual hierarchical design partition separately, and then merges the partitions into a complete netlist for subsequent stages of the compilation flow. When recompiling your design, you can use source code, post-synthesis results, or post-fitting results to preserve satisfactory results for each partition. Refer to "Incremental Compilation Summary" on page 3-7 for more information.

In a team-based environment, part of your design may be incomplete, or it may have been developed by another designer or IP provider. In this scenario, you can add the completed partitions to the design incrementally. Alternatively, other designers or IP providers can develop and optimize partitions independently and the project lead can later integrate the partitions into the top-level design. Refer to "Team-Based Design Flows and IP Delivery" on page 3-6 for more information.

Table 3-1 shows a summary of the impact the Quartus II incremental compilation feature has on compilation results.

Table 3-1. Impact Summary of Using Incremental Compilation (Part 1 of 2)

Characteristic	Impact of Incremental Compilation with Design Partitions
Compilation Time Savings	Typically saves an average of 75% of compilation time for small design changes in large designs when post-fit netlists are preserved; there are savings in both Quartus II Integrated Synthesis and the Fitter. (1)
Performance Preservation	Excellent performance preservation when timing critical paths are contained within a partition, because you can preserve post-fitting information for unchanged partitions.
Node Name Preservation	Preserves post-fitting node names for unchanged partitions.

Table 3–1. Impact Summary of Using Incremental Compilation (Part 2 of 2)

Characteristic	Impact of Incremental Compilation with Design Partitions
Area Changes	The area (logic resource utilization) might increase because cross-boundary optimizations are limited, and placement and register packing are restricted.
f_{MAX} Changes	The design's maximum frequency might be reduced because cross-boundary optimizations are limited. If the design is partitioned and the floorplan location assignments are created appropriately, there might be no negative impact on f_{MAX} .

Note to Table 3–1:

- (1) Quartus II incremental compilation does not reduce processing time for the early "pre-fitter" operations, such as determining pin locations and clock routing, so the feature cannot reduce compilation time if runtime is dominated by those operations.

If you use the incremental compilation feature at any point in your design flow, it is easier to accommodate the guidelines for partitioning a design and creating a floorplan if you start planning for incremental compilation at the beginning of your design cycle.



For more information and recommendations on how to prepare your design to use the Quartus II incremental compilation feature, and how to avoid negative impact on your design results, refer to the *Best Practices for Incremental Compilation Partitions and Floorplan Assignments* chapter in volume 1 of the *Quartus II Handbook*.

Figure 3–1. Quartus II Design Flow Using Incremental Compilation



The diagram in [Figure 3–1](#) shows a top-level partition and two lower-level partitions. If any part of the design changes, Analysis and Synthesis processes the changed partitions and keeps the existing netlists for the unchanged partitions. After completion of Analysis and Synthesis, there is one post-synthesis netlist for each partition.

The Partition Merge step creates a single, complete netlist that consists of post-synthesis netlists, post-fit netlists, and netlists exported from other Quartus II projects, depending on the netlist type that you specify for each partition.

The Fitter then processes the merged netlist, preserves the placement and routing of unchanged partitions, and refits only those partitions that have changed. The Fitter generates the complete netlist for use in future stages of the compilation flow, including timing analysis and programming file generation, which can take place in parallel if more than one processor is enabled for use in the Quartus II software. The Fitter also generates individual netlists for each partition so that the Partition Merge stage can use the post-fit netlist to preserve the placement and routing of a partition, if specified, for future compilations.

If you define partitions, but want to check your compilation results without partitions in a “what if” scenario, you can direct the Compiler to ignore all partitions assignments in your project and compile the design as a “flat” netlist. When you turn on the **Ignore partitions assignments during compilation** option on the **Incremental Compilation** page, the Quartus II software disables all design partition assignments in your project and runs a full compilation ignoring all partition boundaries and netlists. Turning off the **Ignore partitions assignments during compilation** option restores all partition assignments and netlists for subsequent compilations.

❓ For more information on incremental compilation settings, refer to *Incremental Compilation Page* and *Design Partition Properties Dialog Box* in Quartus II Help.

Team-Based Design Flows and IP Delivery

The Quartus II software supports various design flows to enable team-based design and third-party IP delivery. A top-level design can include one or more partitions that are designed or optimized by different designers or IP providers, as well as partitions that will be developed as part of a standard incremental methodology.

In a team-based environment, part of your design may be incomplete because it is being developed elsewhere. The project lead or system architect can create empty placeholders in the top-level design for partitions that are not yet complete. Designers or IP providers can create and verify HDL code separately, and then the project lead later integrates the code into the single top-level Quartus II project. In this scenario, you can add the completed partitions to the design incrementally, however, the design flow allows all design optimization to occur in the top-level design for easiest design integration. Altera recommends using a single Quartus II project whenever possible because using multiple projects can add significant up-front and debugging time to the development cycle.

Alternatively, partition designers can design their partition in a copy of the top-level design or in a separate Quartus II project. Designers export their completed partition as either a post-synthesis netlist or optimized placed and routed netlist, or both, along with assignments such as LogicLock™ regions, as appropriate. The project lead then integrates each design block as a design partition into the top-level design. Altera recommends that designers export and reuse post-synthesis netlists, unless optimized post-fit results are required in the top-level design, to simplify design optimization.

Teams with a bottom-up design approach often want to optimize placement and routing of design partitions independently and may want to create separate Quartus II projects for each partition. However, optimizing design partitions in separate Quartus II projects, and then later integrating the results into a top-level design, can have the following potential drawbacks that require careful planning:

- Achieving timing closure for the full design may be more difficult if you compile partitions independently without information about other partitions in the design. This problem may be avoided by careful timing budgeting and special design rules, such as always registering the ports at the module boundaries.
- Resource budgeting and allocation may be required to avoid resource conflicts and overuse. Creating a floorplan with LogicLock regions is recommended when design partitions are developed independently in separate Quartus II projects.
- Maintaining consistency of assignments and timing constraints can be more difficult if there are separate Quartus II projects. The project lead must ensure that the top-level design and the separate projects are consistent in their assignments.

A unique challenge of team-based design and IP delivery for FPGAs is the fact that the partitions being developed independently must share a common set of resources. To minimize issues that might arise from sharing a common set of resources, you can design partitions within a single Quartus II project or a copy of the top-level design. A common project ensures that designers have a consistent view of the top-level project framework.

For timing-critical partitions being developed and optimized by another designer, it is important that each designer has complete information about the top-level design in order to maintain timing closure during integration, and to obtain the best results. When you want to integrate partitions from separate Quartus II projects, the project lead can perform most of the design planning, and then pass the top-level design constraints to the partition designers. Preferably, partition designers can obtain a copy of the top-level design by checking out the required files from a source control system. Alternatively, the project lead can provide a copy of the top-level project framework, or pass design information using Quartus II-generated design partition scripts. In the case that a third-party designer has no information about the top-level design, developers can export their partition from an independent project if required.

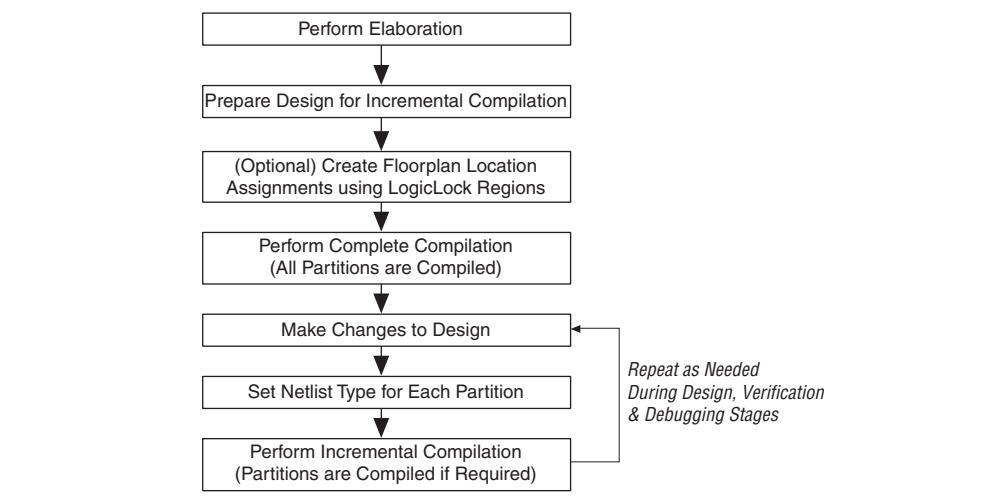
For more information about managing team-based design flows, refer to “Exporting Design Partitions from Separate Quartus II Projects” on page 3-30 and “Project Management—Making the Top-Level Design Available to Other Designers” on page 3-32.

Incremental Compilation Summary

This section provides a summary of the standard incremental compilation design flow and describes how to create design partitions.

Figure 3-2 illustrates the incremental compilation design flow when all partitions are contained in one top-level design.

Figure 3-2. Summary of Standard Incremental Compilation Design Flow



Steps for Incremental Compilation

This section summarizes the steps in an incremental compilation flow; preparing a design to use the incremental compilation feature, and then preserving satisfactory results and performance in subsequent incremental compilations.

- ② For an interactive introduction to implementing an incremental compilation design flow, refer to the **Getting Started Tutorial** on the Help menu in the Quartus II software. For step-by-step instructions on how to use the incremental compilation feature, refer to *Using the Incremental Compilation Design Flow* in Quartus II Help.

Preparing a Design for Incremental Compilation

To begin, elaborate your design, or run any compilation flow (such as a full compilation) that includes the elaboration step. Elaboration is the part of the synthesis process that identifies your design's hierarchy.

Next, designate specific instances in the design hierarchy as design partitions, as described in *"Creating Design Partitions"* on page 3-9.

If required for your design flow, create a floorplan with LogicLock regions location assignments for timing-critical partitions that change with future compilations. Assigning a partition to a physical region on the device can help maintain quality of results and avoid conflicts in certain situations. For more information about LogicLock region assignments, refer to *"Creating a Design Floorplan With LogicLock Regions"* on page 3-48.

Compiling a Design Using Incremental Compilation

The first compilation after making partition assignments is a full compilation, and prepares the design for subsequent incremental compilations. In subsequent compilations of your design, you can preserve satisfactory compilation results and performance of unchanged partitions with the **Netlist Type** setting in the Design Partitions window. The **Netlist Type** setting determines which type of netlist or source file the Partition Merge stage uses in the next incremental compilation. You can choose the Source File, Post-Synthesis netlist, or Post-Fit netlist. For more information about the **Netlist Type** setting, refer to “[Specifying the Level of Results Preservation for Subsequent Compilations](#)” on page 3-25.

Creating Design Partitions

There are several ways to designate a design instance as a design partition. This section provides an overview of tools you can use to create partitions in the Quartus II software. For more information on selecting which design blocks to assign as partitions and how to analyze the quality of your partition assignments, refer to “[Deciding Which Design Blocks Should Be Design Partitions](#)” on page 3-19.

Creating Design Partitions in the Project Navigator

You can right-click an instance in the list under the **Hierarchy** tab in the Project Navigator and use the sub-menu to create and delete design partitions.

- For more information about how to create design partitions in the Quartus II Project Navigator, refer to [Creating Design Partitions](#) in Quartus II Help.

Creating Design Partitions in the Design Partitions Window

The Design Partitions window, available from the Assignments menu, allows you to create, delete, and merge partitions, and is the main window for setting the netlist type to specify the level of results preservation for each partition on subsequent compilations. For information about how to set the netlist type and the available settings, refer to “[Netlist Type for Design Partitions](#)” on page 3-25.

The Design Partitions window also lists recommendations at the bottom of the window with links to the Incremental Compilation Advisor, where you can view additional recommendations about partitions. The **Color** column indicates the color of each partition as it appears in the Design Partition Planner and Chip Planner.

You can right-click a partition in the window to perform various common tasks, such as viewing property information about a partition, including the time and date of the compilation netlists and the partition statistics.

When you create a partition, the Quartus II software automatically generates a name based on the instance name and hierarchy path. You can edit the partition name in the Design Partitions Window so that you avoid referring to them by their hierarchy path, which can sometimes be long. This is especially useful when using command-line commands or assignments, or when you merge partitions to give the partition a meaningful name. Partition names can be from 1 to 1024 characters in length and must be unique. The name can consist of alphanumeric characters and the pipe (|), colon (:), and underscore (_) characters.

- ❓ For more information about how to create and manage design partitions in the Design Partitions window, refer to *Creating Design Partitions* in Quartus II Help.

Creating Design Partitions With the Design Partition Planner

The Design Partition Planner allows you to view design connectivity and hierarchy, and can assist you in creating effective design partitions that follow Altera's guidelines.

The Design Partition Planner displays a visual representation of design connectivity and hierarchy, as well as partitions and entity relationships. You can explore the connectivity between entities in the design, evaluate existing partitions with respect to connectivity between entities, and try new partitioning schemes in "what if" scenarios.

When you extract design blocks from the top-level design and drag them into the Design Partition Planner, connection bundles are drawn between entities, showing the number of connections existing between pairs of entities. In the Design Partition Planner, you can then set extracted design blocks as design partitions.

The Design Partition Planner also has an **Auto-Partition** feature that creates partitions based on the size and connectivity of the hierarchical design blocks.



For more information about how to use the Design Partition Planner, refer to *Using the Design Partition Planner* in Quartus II Help and the *Best Practices for Incremental Compilation Partitions and Floorplan Assignments* chapter in volume 1 of the *Quartus II Handbook*.

Creating Design Partitions With Tcl Scripting

You can also create partitions with Tcl scripting commands. For more information about the command-line and scripting flow, refer to "Scripting Support" on page 3-57.

Automatically-Generated Partitions

The Compiler creates some partitions automatically as part of the compilation process, which appear in some post-compilation reports. For example, the `sld_hub` partition is created for tools that use JTAG hub connections, such as the SignalTap II Logic Analyzer. The `hard_block` partition is created to contain certain "hard" or dedicated logic blocks in the device that are implemented in a separate partition so that they can be shared throughout the design.

Common Design Scenarios Using Incremental Compilation

This section provides recommended applications of the incremental compilation flow after you have set up your design with partitions for incremental compilation as described in, "Steps for Incremental Compilation" on page 3-8.

This section contains the following design scenarios:

- "Reducing Compilation Time When Changing Source Files for One Partition" on page 3-11
- "Optimizing a Timing-Critical Partition" on page 3-11

- “Adding Design Logic Incrementally or Working With an Incomplete Design” on page 3-12
- “Debugging Incrementally With the SignalTap II Logic Analyzer” on page 3-13

Reducing Compilation Time When Changing Source Files for One Partition

Scenario background: You set up your design to include partitions for several of the major design blocks, and now you have just performed a lengthy compilation of the entire design. An error is found in the HDL source file for one partition and it is being fixed. Because the design is currently meeting timing requirements, and the fix is not expected to affect timing performance, it makes sense to compile only the affected partition and preserve the rest of the design.

Use the flow in this example to update the source file in one partition without having to recompile the other parts of the design. To reduce the compilation time, instruct the software to reuse the post-fit netlists for the unchanged partitions. This flow also preserves the performance of these blocks, which reduces additional timing closure efforts.

Perform the following steps to update a single source file:

1. Apply and save the fix to the HDL source file.
2. On the Assignments menu, open the **Design Partitions** window.
3. Change the netlist type of each partition, including the top-level entity, to **Post-Fit** to preserve as much as possible for the next compilation.



The Quartus II software recompiles partitions by default when changes are detected in a source file. You can refer to the Partition Dependent Files table in the Analysis and Synthesis report to determine which partitions were recompiled. If you change an assignment but do not change the logic in a source file, you can set the netlist type to **Source File** for that partition to instruct the software to recompile the partition's source design files and its assignments.



For more information about the Analysis and Synthesis report, refer to *List of Compilation and Simulation Reports* in Quartus II Help.

4. Click **Start Compilation** to incrementally compile the fixed HDL code. This compilation should take much less time than the initial full compilation.
5. Simulate the design to ensure that the error is fixed, and use the TimeQuest Timing Analyzer report to ensure that timing results have not degraded.

Optimizing a Timing-Critical Partition

Scenario background: You have just performed a lengthy full compilation of a design that consists of multiple partitions. The TimeQuest Timing Analyzer reports that the clock timing requirement is not met, and you have to optimize one particular partition. You want to try optimization techniques such as raising the Placement Effort Multiplier, enabling Physical Synthesis, and running the Design Space Explorer. Because these techniques all involve significant compilation time, you should apply them to only the partition in question.

Use the flow in this example to optimize the results of one partition when the other partitions in the design have already met their requirements. You can use this flow iteratively to lock down the performance of one partition, and then move on to optimization of another partition.

Perform the following steps to preserve the results for partitions that meet their timing requirements, and to recompile a timing-critical partition with new optimization settings:

1. Open the **Design Partitions** window.
2. For the partition in question, set the netlist type to **Source File**.



If you change a setting that affects only the Fitter, you can save additional compilation time by setting the netlist type to **Post-Synthesis** to reuse the synthesis results and refit the partition.

3. For the remaining partitions (including the top-level entity), set the netlist type to **Post-Fit**.



You can optionally set the **Fitter Preservation Level** on the **Advanced** tab in the **Design Partitions Properties** dialog box to **Placement** to allow for the most flexibility during routing.

4. Apply the desired optimization settings.
5. Click **Start Compilation** to perform incremental compilation on the design with the new settings. During this compilation, the Partition Merge stage automatically merges the critical partition's new synthesis netlist with the post-fit netlists of the remaining partitions. The Fitter then refits only the required partition. Because the effort is reduced as compared to the initial full compilation, the compilation time is also reduced.

To use the Design Space Explorer, perform the following steps:

1. Repeat steps 1–3 of the previous procedure.
2. Save the project and run the Design Space Explorer.

Adding Design Logic Incrementally or Working With an Incomplete Design

Scenario background: You have one or more partitions that are known to be timing-critical in your full design. You want to focus on developing and optimizing this subset of the design first, before adding the rest of the design logic.

Use this flow to compile a timing-critical partition or partitions in isolation, optionally with extra optimizations turned on. After timing closure is achieved for the critical logic, you can preserve its content and placement and compile the remaining partitions with normal or reduced optimization levels. For example, you may want to compile an IP block that comes with instructions to perform optimization before you incorporate the rest of your custom logic.

To implement this design flow, perform the following steps:

1. Partition the design and create floorplan location assignments. For best results, ensure that the top-level design includes the entire project framework, even if some parts of the design are incomplete and are represented by an empty wrapper file.
2. For the partitions to be compiled first, in the Design Partitions window, set the netlist type to **Source File**.
3. For the remaining partitions, set the netlist type to **Empty**.
4. To compile with the desired optimizations turned on, click **Start Compilation**.
5. Check the Timing Analyzer reports to ensure that timing requirements are met. If so, proceed to step 6. Otherwise, repeat steps 4 and 5 until the requirements are met.
6. In the Design Partitions window, set the netlist type to **Post-Fit** for the first partitions. You can set the **Fitter Preservation Level** on the **Advanced** tab in the **Design Partitions Properties** dialog box to **Placement** to allow more flexibility during routing if exact placement and routing preservation is not required.
7. Change the netlist type from **Empty** to **Source File** for the remaining partitions, and ensure that the completed source files are added to the project.
8. Set the appropriate level of optimizations and compile the design. Changing the optimizations at this point does not affect any fitted partitions, because each partition has its netlist type set to **Post-Fit**.
9. Check the Timing Analyzer reports to ensure that timing requirements are met. If not, make design or option changes and repeat step 8 and step 9 until the requirements are met.



The flow in this example is similar to design flows in which a module is implemented separately and is later merged into the top-level, such as in the team-based design flow described in “[Designing in a Team-Based Environment](#)” on page 3-42. Generally, optimization in this flow works only if each critical path is contained within a single partition due to the effects described in “[Deciding Which Design Blocks Should Be Design Partitions](#)” on page 3-19. Ensure that if there are any partitions representing a design file that is missing from the project, you create a placeholder wrapper file to define the port interface. For more information, refer to “[Empty Partitions](#)” on page 3-32.

Debugging Incrementally With the SignalTap II Logic Analyzer

Scenario background: Your design is not functioning as expected, and you want to debug the design using the SignalTap II Logic Analyzer. To maintain reduced compilation times and to ensure that you do not negatively affect the current version of your design, you want to preserve the synthesis and fitting results and add the SignalTap II Logic Analyzer to your design without recompiling the source code.

Use this flow to reduce compilation times when you add the logic analyzer to debug your design, or when you want to modify the configuration of the SignalTap II File without modifying your design logic or its placement.

It is not necessary to create design partitions in order to use the SignalTap II incremental compilation feature. The SignalTap II Logic Analyzer acts as its own separate design partition.

Perform the following steps to use the SignalTap II Logic Analyzer in an incremental compilation flow:

1. Open the Design Partitions window.
2. Set the netlist type to **Post-fit** for all partitions to preserve their placement.



The netlist type for the top-level partition defaults to **Source File**, so be sure to change this “Top” partition in addition to any design partitions that you have created.

3. If you have not already compiled the design with the current set of partitions, perform a full compilation. If the design has already been compiled with the current set of partitions, the design is ready to add the SignalTap II Logic Analyzer.
4. Set up your SignalTap II File using the **post-fitting** filter in the **Node Finder** to add signals for logic analysis. This allows the Fitter to add the SignalTap II logic to the post-fit netlist without modifying the design results.

To add signals from the pre-synthesis netlist, set the partition’s netlist type to **Source File** and use the **pre-synthesis** filter in the **Node Finder**. This allows the software to resynthesize the partition and to tap directly to the pre-synthesis node names that you choose. In this case, the partition is resynthesized and refit, so the placement is typically different from previous fitting results.



For more information about setting up the SignalTap II Logic Analyzer, refer to the *Design Debugging Using the SignalTap II Embedded Logic Analyzer* chapter in volume 3 of the *Quartus II Handbook*.

Functional Safety IP Implementation

In functional safety designs, recertification is required when logic is modified in safety or non-safety areas of the design. Recertification is required because the FPGA programming file has changed. You can reduce the amount of required recertification if you use the safety/non-safety separation flow in the Quartus II software. By partitioning your safety IP from non-safety related logic, you ensure that the safety critical areas of the design remain the same when the non-safety areas in your design are modified. The safety-critical areas remain the same at the bit level.

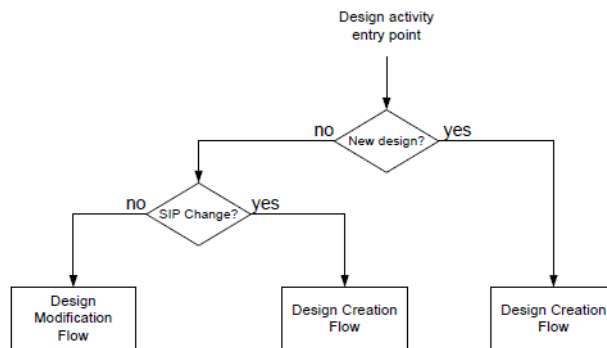
IEC61508 Compliance

The Quartus II software can partition your design into safety partitions and non-safety partitions, but the Quartus II software does not perform any online safety-related functionality. A bitstream is generated by the Quartus II software that performs the safety functions and for the purposed of compliance with IEC61508, the Quartus II software should be considered as an offline support tool.

Functional Safety Separation Flow

The functional safety separation flow consists of two separate work flows. The design creation flow (DCF) and the design modification flow (DMF) both leverage incremental compilation, but the two flows have different use-case scenarios.

Figure 3–3. Functional Safety Separation Flow



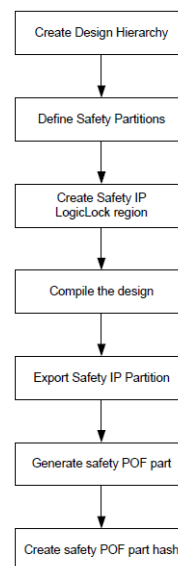
Design Creation Flow

The design creation flow delineates the necessary steps for initial design creation in a way that allows modifications to be made in your design. Some of the steps are architectural constraints and the remaining steps are steps that you need to perform in the Quartus II software. You use DCF for the first pass certification of your product.



When you make modifications to the safety IP in your design, you are required to use the design creation flow.

Figure 3–4. Design Creation Flow



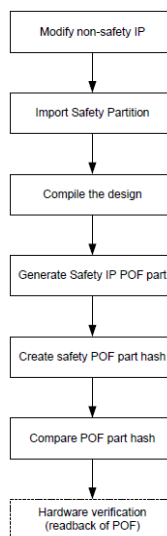
Design Modification Flow

The design modification flow delineates the necessary steps to make modifications to the non-safety IP in your design. This flow ensures that the previously compiled Safety IP (SIP) that is used in the project remains unchanged when non-safety IP (NSIP) changes are made or compiled.



You can only use the design modification flow after your design has been qualified in the design creation flow.

Figure 3-5. Design Modification Flow



How to Turn On the Functional Safety Separation Flow

Every safety related IP component in your design should be implemented in a partition(s) so the SIPs are protected from recompilation. The global assignment `PARTITION_ENABLE_STRICT_PRESERVATION` is used to identify SIP in your design.

```

■ set_global_assignment -name PARTITION_ENABLE_STRICT_PRESERVATION
  <ON/OFF> -section_id <partition_name>
  
```

When this global assignment is designated as ON for a partition, the partition is protected from recompilation, exported as a SIP, and included into the SIP POF mask. Specifying the value as ON for any partition turns on the functional safety separation flow.

When this global assignment is designated as OFF, the partition is considered as part of the NSIP or as not having a `PARTITION_ENABLE_STRICT_PRESERVATION` assignment at all. Logic that is not assigned to a partition is considered as part of the top partition and treated as non-safety logic.



Only partitions and I/O pins can be assigned to SIP.

A partition assigned to SIP can contain safety logic only. If the parent partition is assigned to a SIP, then all the child partitions for this parent partition is considered as part of the SIP. If a child partition is not specified explicitly as a SIP, a critical warning is issued to notify you that the child partition is treated as part of a SIP.

A design can contain several SIPs. All the partitions containing logic that implements a single SIP function should belong with the same top level parent partition.

The functional safety separation flow supports Cyclone IV and Cyclone V device families.

You can also turn on the functional safety separation flow from the **Design Partition Properties** dialog box.

When the functional safety separation flow is active, you can view which partitions in your design have the Strict Preservation property turned on. The **Design Partition Window** displays a on or off value for SIP in your design.

- ❓ For more information about the *Design Partition Properties* dialog box and the *Design Partitions Window*, refer to the Quartus II Help.

Preservation of Device Resources

The preservation of the partition's netlist atoms and the atoms placement and routing, in the design modification flow, is done by setting the netlist type to Post-fit with the Fitter preservation level set to Placement and Routing Preserved.

Preservation of Placement in the Device with LogicLock

In order to fix the SIP logic into specific areas of the device, you should define LogicLock regions. By using preserved LogicLock regions, device placement is reserved for the SIP to prevent NSIP logic from being placed into the unused resources of the SIP region. You establish a fixed size and origin to ensure location preservation. You need to use LogicLock to ensure a valid SIP POF mask is generated, but the SIP POF mask gets generated when you turn on the functional safety separation flow. The POF comparison tool for functional safety can check that the safety region is unchanged between compiles. A LogicLock region assigned to a SIP can only contain safety IP logic.

Assigning I/O Pins

You can use a global assignment to specify that a pin is assigned to a SIP.

```
set_instance_assignment ENABLE_STRICT_PRESERVATION ON/OFF - to=<hpath> -  
section_id <region_name>
```

- <hpath> refers to an I/O pin (pad).
- <region_name> refers to the top level SIP partition name.

A value of ON indicates that the pin is a safety pin that should be preserved along with the SIP. A value of OFF indicates that the pin that connects up to the SIP, should be treated as a non-safety pin, and is not preserved along with the SIP.

All the pins that connect up to a SIP should have an explicit assignment.

An error is reported if a pin that connects up the SIP does not have an assignment or a pin does not connect up to the specified <region_name>.

If an IO_REG group contains a pin that is assigned to a SIP, then all the pins in the IO_REG group are reserved for this SIP. All pins in the IO_REG group need to be assigned to the same SIP and none of the pins in the group can be assigned to non-safety signals.

General Guidelines for Implementation

- An internal clock source, such as a PLL, should be implemented in a safe partition.
- An I/O pin driving the external clock should be indicated as a safety pin.
- To export a SIP containing several partitions, the top level partition for the SIP should be exported. A SIP containing several partitions is flattened and converted into a single partition during export. This hierarchical SIP is flattened to ensure bit-level settings are preserved.
- Hard blocks implemented in a safe partition needs to stay with the safe partition.

Reports for SIP

When you have the functional safety separation flow turned on, the Quartus II software displays SIP and NSIP information in the Fitter report.

Fitter Report

The Fitter report includes information for each SIP and the respective partition and I/O usage. The report contains the following information:

- Partition name (with the name of the top level SIP partition used as the SIP name)
- Number of safety/non-safety inputs to the partitions
- Number of safety/non-safety outputs to the partitions
- LogicLock region names along with size and locations for the regions
- I/O pins used for the respective SIP in your design
- Safety related error messages

SIP Partial Bitstream Generation

The Programmer generates a bitstream file containing only the bits for a SIP. This partial preserved bitstream (PPB) file is for the SIP region mask. The command lines to generate the partial bitstream file are the following:

- `quartus_cpf --gen_ppb safe1.psm design.sof safe1.rbf.ppb`
- `quartus_cpf -c safe1.psm safe1.rbf.ppb`

The PPB file is generated in two steps.

1. Generation of partial SOF.
2. Generation of PPB file using the partial SOF.

POF Comparison Tool for Verification

There is a separate safe/non-safe partitioning verification tool that is licensed to safety users. Along with the PPB file, a MD5 hash signature file is generated. The MD5 hash signature can be used for verification. For more detailed verification, the POF comparison tool should be used. This POF comparison tool is available in the Altera Functional Safety Data Package.

Deciding Which Design Blocks Should Be Design Partitions

The incremental compilation design flow requires more planning than flat compilations. For example, you might have to structure your source code or design hierarchy to ensure that logic is grouped correctly for optimization.

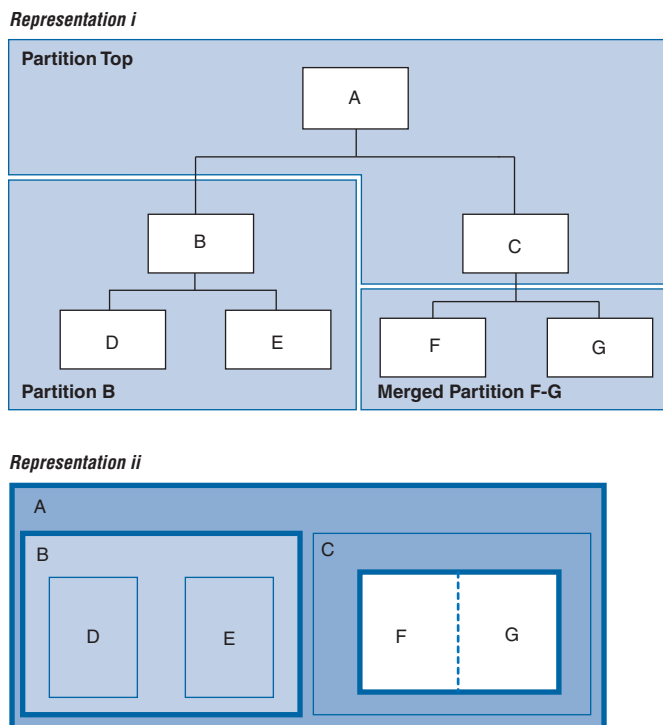
It is a common design practice to create modular or hierarchical designs in which you develop each design entity separately, and then instantiate them in a higher-level entity, forming a complete design. The Quartus II software does not automatically consider each design entity or instance to be a design partition for incremental compilation; instead, you must designate one or more design hierarchies below the top-level project as a design partition. Creating partitions might prevent the Compiler from performing optimizations across partition boundaries, as discussed in [“Impact of Design Partitions on Design Optimization”](#) on page 3-20. However, this allows for separate synthesis and placement for each partition, making incremental compilation possible.

Partitions must have the same boundaries as hierarchical blocks in the design because a partition cannot be a portion of the logic within a hierarchical entity. You can merge partitions that have the same immediate parent partition to create a single partition that includes more than one hierarchical entity in the design. When you declare a partition, every hierarchical instance within that partition becomes part of the same partition. You can create new partitions for hierarchical instances within an existing partition, in which case the instances within the new partition are no longer included in the higher-level partition, as described in the following example.

In [Figure 3-6](#), a complete design is made up of instances A, B, C, D, E, F, and G. The shaded boxes in Representation i indicate design partitions in a “tree” representation of the hierarchy. In Representation ii, the lower-level instances are represented inside the higher-level instances, and the partitions are illustrated with different colored shading. The top-level partition, called “Top”, automatically contains the top-level entity in the design, and contains any logic not defined as part of another partition. The design file for the top level may be just a wrapper for the hierarchical instances

below it, or it may contain its own logic. In this example, partition **B** contains the logic in instances **B**, **D**, and **E**. Entities **F** and **G** were first identified as separate partitions, and then merged together to create a partition **F-G**. The partition for the top-level entity **A**, called “Top”, includes the logic in one of its lower-level instances, **C**, because **C** was not defined as part of any other partition.

Figure 3–6. Partitions in a Hierarchical Design



You can create partition assignments to any design instance. The instance can be defined in HDL or schematic design, or come from a third-party synthesis tool as a VQM or EDIF netlist instance.

To take advantage of incremental compilation when source files change, create separate design files for each partition. If you define two different entities as separate partitions but they are in the same design file, you cannot maintain incremental compilation because the software would have to recompile both partitions if you changed either entity in the design file. Similarly, if two partitions rely on the same lower-level entity definition, changes in that lower-level affect both partitions.

The remainder of this section provides information to help you choose which design blocks you should assign as partitions.

Impact of Design Partitions on Design Optimization

The boundaries of your design partitions can impact the design’s quality of results. Creating partitions might prevent the Compiler from performing logic optimizations across partition boundaries, which allows the software to synthesize and place each partition separately in an incremental flow. Therefore, consider partitioning guidelines to help reduce the effect of partition boundaries.

Whenever possible, register all inputs and outputs of each partition. This helps avoid any delay penalty on signals that cross partition boundaries and keeps each register-to-register timing path within one partition for optimization. In addition, minimize the number of paths that cross partition boundaries. If there are timing-critical paths that cross partition boundaries, rework the partitions to avoid these inter-partition paths. Including as many of the timing-critical connections as possible inside a partition allows you to effectively apply optimizations to that partition to improve timing, while leaving the rest of the design unchanged.

Avoid constant partition inputs and outputs. You can also merge two or more partitions to allow cross-boundary optimizations for paths that cross between the partitions, as long as the partitions have the same parent partition. Merging related logic from different hierarchy blocks into one partition can be useful if you cannot change the design hierarchy to accommodate partition assignments.

The Design Partition Planner can help you create good assignments, as described in “Creating Design Partitions” on page 3-9. Refer to “Partition Statistics Reports” on page 3-23 for information about the number of I/O connections and how many are unregistered or driven by a constant value. For information on timing reports and additional design guidelines, refer to “Partition Timing Reports” on page 3-24 and “Incremental Compilation Advisor” on page 3-24.

If critical timing paths cross partition boundaries, you can perform timing budgeting and make timing assignments to constrain the logic in each partition so that the entire timing path meets its requirements. In addition, because each partition is optimized independently during synthesis, you may have to perform resource allocation to ensure that each partition uses an appropriate number of device resources. If design partitions are compiled in separate Quartus II projects, there may be conflicts related to global routing resources for clock signals when the design is integrated into the top-level design. You can use the Global Signal logic option to specify which clocks should use global or regional routing, use the ALTCLK_CTRL megafunction to instantiate a clock control block and connect it appropriately in both the partitions being developed in separate Quartus II projects, or find the compiler-generated clock control node in your design and make clock control location assignments in the Assignment Editor.

Turning On Supported Cross-boundary Optimizations

You can improve the optimizations performed between design partitions by turning on supported cross-boundary optimizations. These optimizations are turned on a per partition basis and you can select the optimizations as individual assignments. This allows the cross-boundary optimization feature to give you more control over the optimizations that work best for your design. You can turn on the cross-boundary optimizations for your design partitions on the **Advanced** tab of the **Design Partition Properties** dialog box. Once you change the optimization settings, the Quartus II software recompiles your partition from source automatically. Cross-boundary optimizations include the following: propagate constants, propagate inversions on partition inputs, merge inputs fed by a common source, merge electrically equivalent bidirectional pins, absorb internal paths, and remove logic connected to dangling outputs.

Cross-boundary optimizations are implemented top-down from the parent partition into the child partition, but not vice-versa. Also, cross-boundary optimizations cannot be enabled for partitions that allow multiple personas (partial reconfiguration partitions).

- For more information about cross-boundary optimizations in the Quartus II software, refer to *Design Partition Properties Dialog Box* in Quartus II Help.



For more partitioning guidelines and specific recommendations for fixing common design issues, as well as information on resource allocation, global signal usage, and timing budgeting, refer to the *Best Practices for Incremental Compilation Partitions and Floorplan Assignments* chapter in volume 1 of the *Quartus II Handbook*.

Design Partition Assignments Compared to Physical Placement Assignments

Design partitions for incremental compilation are logical partitions, which is different from physical placement assignments in the device floorplan. A logical design partition does not refer to a physical area of the device and does not directly control the placement of instances. A logical design partition sets up a virtual boundary between design hierarchies so that each is compiled separately, preventing logical optimizations from occurring between them. When the software compiles the design source code, the logic in each partition can be placed anywhere in the device unless you make additional placement assignments.

If you preserve the compilation results using a Post-Fit netlist, it is not necessary for you to back-annotate or make any location assignments for specific logic nodes. You should not use the incremental compilation and logic placement back-annotation features in the same Quartus II project. The incremental compilation feature does not use placement “assignments” to preserve placement results; it simply reuses the netlist database that includes the placement information.

You can assign design partitions to physical regions in the device floorplan using LogicLock region assignments. In the Quartus II software, LogicLock regions are used to constrain blocks of a design to a particular region of the device. Altera recommends using LogicLock regions for timing-critical design blocks that will change in subsequent compilations, or to improve the quality of results and avoid placement conflicts in some cases. Creating floorplan location assignments for design partitions using LogicLock regions is discussed in “*Creating a Design Floorplan With LogicLock Regions*” on page 3–48.



For more information about when and why to create a design floorplan, refer to the *Best Practices for Incremental Compilation Partitions and Floorplan Assignments* chapter in volume 1 of the *Quartus II Handbook*.

Using Partitions With Third-Party Synthesis Tools

If you are using a third-party synthesis tool, set up your tool to create a separate VQM or EDIF netlist for each hierarchical partition. In the Quartus II software, assign the top-level entity from each netlist to be a design partition. The VQM or EDIF netlist file is treated as the source file for the partition in the Quartus II software.

Synopsys Synplify Pro/Premier and Mentor Graphics Precision RTL Plus

The Synplify Pro and Synplify Premier software include the MultiPoint synthesis feature to perform incremental synthesis for each design block assigned as a Compile Point in the user interface or a script. The Precision RTL Plus software includes an incremental synthesis feature that performs block-based synthesis based on Partition assignments in the source HDL code. These features provide automated block-based incremental synthesis flows and create different output netlist files for each block when set up for an Altera device.

Using incremental synthesis within your synthesis tool ensures that only those sections of a design that have been updated are resynthesized when the design is compiled, reducing synthesis run time and preserving the results for the unchanged blocks. You can change and resynthesize one section of a design without affecting other sections of the design.



For more information about these incremental synthesis flows, refer to your tool vendor's documentation, or the *Synopsys Synplify Support* chapter or *Mentor Graphics Precision Synthesis Support* chapter in volume 1 of the *Quartus II Handbook*.

Other Synthesis Tools

You can also partition your design and create different netlist files manually with the basic Synplify software (non-Pro/Premier), the basic Precision RTL software (non-Plus), or any other supported synthesis tool by creating a separate project or implementation for each partition, including the top level. Set up each higher-level project to instantiate the lower-level VQM/EDIF netlists as black boxes. Synplify, Precision, and most synthesis tools automatically treat a design block as a black box if the logic definition is missing from the project. Each tool also includes options or attributes to specify that the design block should be treated as a black box, which you can use to avoid warnings about the missing logic.

Assessing Partition Quality

The Quartus II software provides various tools to assess the quality of your assigned design partitions. You can take advantage of these tools to assess your partition quality, and use the information to improve your design or assignments as required to achieve the best results.

Partition Statistics Reports

After compilation, you can view statistics about design partitions in the Partition Merge Partition Statistics report, and on the **Statistics** tab in the **Design Partitions Properties** dialog box.

The Partition Merge Partition Statistics report lists statistics about each partition. The statistics for each partition (each row in the table) include the number of logic cells it contains, as well as the number of input and output pins it contains, and how many are registered or unconnected. This report is useful when optimizing your design partitions, ensuring that the partitions meet the guidelines presented in the *Best Practices for Incremental Compilation Partitions and Floorplan Assignments* chapter in volume 1 of the *Quartus II Handbook*.

You can also view post-compilation statistics about the resource usage and port connections for a particular partition on the **Statistics** tab in the **Design Partition Properties** dialog box.

Partition Timing Reports

You can generate a Partition Timing Overview report and a Partition Timing Details report by clicking **Report Partitions** in the Tasks pane in the TimeQuest Timing Analyzer, or using the `report_partitions` Tcl command.

The Partition Timing Overview report shows the total number of failing paths for each partition and the worst-case slack for any path involving the partition.

The Partition Timing Details report shows the number of failing partition-to-partition paths and worst-case slack for partition-to-partition paths, to provide a more detailed breakdown of where the critical paths in the design are located with respect to design partitions.

Incremental Compilation Advisor

You can use the Incremental Compilation Advisor to check that your design follows Altera's recommendations for creating design partitions and floorplan location assignments.

Recommendations are split into **General Recommendations**, **Timing Recommendations**, and **Team-Based Design Recommendations** that apply to design flows in which partitions are compiled independently in separate Quartus II projects before being integrated into the top-level design. Each recommendation provides an explanation, describes the effect of the recommendation, and provides the action required to make a suggested change. In some cases, there is a link to the appropriate Quartus II settings page where you can make a suggested change to assignments or settings. For some items, if your design does not follow the recommendation, the **Check Recommendations** operation creates a table that lists any nodes or paths in your design that could be improved. The relevant timing-independent recommendations for the design are also listed in the Design Partitions window and the LogicLock Regions window.

To verify that your design follows the recommendations, go to the **Timing Independent Recommendations** page or the **Timing Dependent Recommendations** page, and then click **Check Recommendations**. For large designs, these operations can take a few minutes.

After you perform a check operation, symbols appear next to each recommendation to indicate whether the design or project setting follows the recommendations, or if some or all of the design or project settings do not follow the recommendations. Following these recommendations is not mandatory to use the incremental compilation feature. The recommendations are most important to ensure good results for timing-critical partitions.

For some items in the Advisor, if your design does not follow the recommendation, the **Check Recommendations** operation lists any parts of the design that could be improved. For example, if not all of the partition I/O ports follow the **Register All Non-Global Ports** recommendation, the advisor displays a list of unregistered ports with the partition name and the node name associated with the port.

When the advisor provides a list of nodes, you can right-click a node, and then click **Locate** to cross-probe to other Quartus II features, such as the RTL Viewer, Chip Planner, or the design source code in the text editor.



Opening a new TimeQuest report resets the Incremental Compilation Advisor results, so you must rerun the Check Recommendations process.

Specifying the Level of Results Preservation for Subsequent Compilations

As introduced in “Incremental Compilation Summary” on page 3-7 and “Common Design Scenarios Using Incremental Compilation” on page 3-10, the netlist type of each design partition allows you to specify the level of results preservation. The netlist type determines which type of netlist or source file the Partition Merge stage uses in the next incremental compilation.

When you choose to preserve a post-fit compilation netlist, the default level of Fitter preservation is the highest degree of placement and routing preservation supported by the device family. The advanced Fitter Preservation Level setting allows you to specify the amount of information that you want to preserve from the post-fit netlist file.

Netlist Type for Design Partitions

Before starting a new compilation, ensure that the appropriate netlist type is set for each partition to preserve the desired level of compilation results. Table 3-2 describes the settings for the netlist type, explains the behavior of the Quartus II software for each setting, and provides guidance on when to use each setting.

Table 3-2. Partition Netlist Type Settings (Part 1 of 2)

Netlist Type	Quartus II Software Behavior for Partition During Compilation
Source File	<p>Always compiles the partition using the associated design source file(s). ⁽¹⁾</p> <p>Use this netlist type to recompile a partition from the source code using new synthesis or Fitter settings.</p>
Post-Synthesis	<p>Preserves post-synthesis results for the partition and reuses the post-synthesis netlist when the following conditions are true:</p> <ul style="list-style-type: none"> ■ A post-synthesis netlist is available from a previous synthesis. ■ No change that initiates an automatic resynthesis has been made to the partition since the previous synthesis. ⁽²⁾ For details, refer to “What Changes Initiate the Automatic Resynthesis of a Partition?” on page 3-28. <p>Compiles the partition from the source files if resynthesis is initiated or if a post-synthesis netlist is not available. ⁽¹⁾</p> <p>Use this netlist type to preserve the synthesis results unless you make design changes, but allow the Fitter to refit the partition using any new Fitter settings.</p>

Table 3–2. Partition Netlist Type Settings (Part 2 of 2)

Netlist Type	Quartus II Software Behavior for Partition During Compilation
Post-Fit	<p>Preserves post-fit results for the partition and reuses the post-fit netlist when the following conditions are true:</p> <ul style="list-style-type: none"> ■ A post-fit netlist is available from a previous fitting. ■ No change that initiates an automatic resynthesis has been made to the partition since the previous fitting. ⁽²⁾ For details, refer to “What Changes Initiate the Automatic Resynthesis of a Partition?” on page 3–28. <p>When a post-fit netlist is not available, the software reuses the post-synthesis netlist if it is available, or otherwise compiles from the source files. Compiles the partition from the source files if resynthesis is initiated. ⁽¹⁾</p> <p>The Fitter Preservation Level specifies what level of information is preserved from the post-fit netlist. For details, refer to “Fitter Preservation Level for Design Partitions” on page 3–26.</p> <p>Assignment changes, such as Fitter optimization settings, do not cause a partition set to Post-Fit to recompile.</p>
Empty	<p>Uses an empty placeholder netlist for the partition. The partition's port interface information is required during Analysis and Synthesis to connect the partition correctly to other logic and partitions in the design, and peripheral nodes in the source file including pins and PLLs are preserved to help connect the empty partition to the rest of the design and preserve timing of any lower-level non-empty partitions within empty partitions. If the source file is not available, you can create a wrapper file that defines the design block and specifies the input, output, and bidirectional ports. In Verilog HDL: a module declaration, and in VHDL: an entity and architecture declaration.</p> <p>You can use this netlist type to skip the compilation of a partition that is incomplete or missing from the top-level design. You can also set an empty partition if you want to compile only some partitions in the design, such as to optimize the placement of a timing-critical block such as an IP core before incorporating other design logic, or if the compilation time is large for one partition and you want to exclude it.</p> <p>If the project database includes a previously generated post-synthesis or post-fit netlist for an unchanged Empty partition, you can set the netlist type from Empty directly to Post-Synthesis or Post-Fit and the software reuses the previous netlist information without recompiling from the source files.</p>

Notes to Table 3–2:

- (1) If you use Rapid Recompile, the Quartus II software might not recompile the entire partition from the source code as described in this table; it will reuse compatible results if there have been only small changes to the logic in the partition. Refer to “Incremental Capabilities Available When A Design Has No Partitions” on page 3–2 for more information.
- (2) You can turn on the **Ignore changes in source files and strictly use the specified netlist, if available** option on the **Advanced** tab in the **Design Partitions Properties** dialog box to specify whether the Compiler should ignore source file changes when deciding whether to recompile the partition.


Fitter Preservation Level for Design Partitions

The default Fitter Preservation Level for partitions with a **Post-Fit** netlist type is the highest level of preservation available for the target device family and provides the most compilation time reduction.

You can change the advanced Fitter Preservation Level setting to provide more flexibility in the Fitter during placement and routing. You can set the Fitter Preservation Level on the **Advanced** tab in the **Design Partitions Properties** dialog box. [Table 3-3](#) describes the Fitter Preservation Level settings.

Table 3-3. Fitter Preservation Level Settings

Fitter Preservation Level	Quartus II Behavior for Partition During Compilation
Placement and Routing	Preserves the design partition's netlist atoms and their placement and routing. This setting reduces compilation times compared to Placement only, but provides less flexibility to the router to make changes if there are changes in other parts of the design. By default, the Fitter preserves the usage of high-speed programmable power tiles contained within the selected partition, for devices that support high-speed and low-power tiles. You can turn off the Preserve high-speed tiles when preserving placement and routing option on the Advanced tab in the Design Partitions Properties dialog box.
Placement	Preserves the netlist atoms and their placement in the design partition. Reroutes the design partition and does not preserve high-speed power tile usage.
Netlist Only	Preserves the netlist atoms of the design partition, but replaces and reroutes the design partition. A post-fit netlist with the atoms preserved can be different than the Post-Synthesis netlist because it contains Fitter optimizations; for example, Physical Synthesis changes made during a previous Fitting. You can use this setting to: <ul style="list-style-type: none"> ■ Preserve Fitter optimizations but allow the software to perform placement and routing again. ■ Reapply certain Fitter optimizations that would otherwise be impossible when the placement is locked down. ■ Resolve resource conflicts between two imported partitions.

 For more information about how to set the **Netlist Type** and **Fitter Preservation Level** settings in the Quartus II software, refer to *Setting the Netlist Type and Fitter Preservation Level for Design Partitions* in Quartus II Help.

Where Are the Netlist Databases Saved?

The incremental compilation database folder (**\incremental_db**) includes all the netlist information from previous compilations. To avoid unnecessary recompilations, these database files must not be altered or deleted.

If you archive or reproduce the project in another location, you can use a Quartus II Archive File (**.qar**). Include the incremental compilation database files to preserve post-synthesis or post-fit compilation results. For more information, refer to *“Using Incremental Compilation With Quartus II Archive Files”* on page 3-52.

To manually create a project archive that preserves compilation results without keeping the incremental compilation database, you can keep all source and settings files, and create and save a Quartus II Settings File (**.qxp**) for each partition in the design that will be integrated into the top-level design. For more information about how to create a **.qxp** for a partition within your design, refer to *“Exporting Design Partitions from Separate Quartus II Projects”* on page 3-30.

Deleting Netlists

You can choose to abandon all levels of results preservation and remove all netlists that exist for a particular partition with the **Delete Netlists** command in the Design Partitions window. When you delete netlists for a partition, the partition is compiled using the associated design source file(s) in the next compilation. Resetting the netlist type for a partition to **Source** would have the same effect, though the netlists would not be permanently deleted and would be available for use in subsequent compilations. For an imported partition, the **Delete Netlists** command also optionally allows you to remove the imported **.qxp**.

What Changes Initiate the Automatic Resynthesis of a Partition?

A partition is synthesized from its source files if there is no post-synthesis netlist available from a previous synthesis, or if the netlist type is set to **Source File**. Additionally, certain changes to a partition initiate an automatic resynthesis of the partition when the netlist type is **Post-Synthesis** or **Post-Fit**. The software resynthesizes the partition in these cases to ensure that the design description matches the post-place-and-route programming files. If you do not want resynthesis to occur automatically, refer to [“Forcing Use of the Compilation Netlist When a Partition has Changed”](#) on page 3-30.

The following list explains the changes that initiate a partition’s automatic resynthesis when the netlist type is set to **Post-Synthesis** or **Post-Fit**:

- The device family setting has changed.
- Any dependent source design file has changed. For more information, refer to [“Resynthesis Due to Source Code Changes”](#) on page 3-29.
- The partition boundary was changed by an addition, removal, or change to the port boundaries of a partition (for example, a new partition has been defined for a lower-level instance within this partition).
- A dependent source file was compiled into a different library (so it has a different **-library** argument).
- A dependent source file was added or removed; that is, the partition depends on a different set of source files.
- The partition’s root instance has a different entity binding. In VHDL, an instance may be bound to a specific entity and architecture. If the target entity or architecture changes, it triggers resynthesis.
- The partition has different parameters on its root hierarchy or on an internal AHDL hierarchy (AHDL automatically inherits parameters from its parent hierarchies). This occurs if you modified the parameters on the hierarchy directly, or if you modified them indirectly by changing the parameters in a parent design hierarchy.
- You have moved the project and compiled database between a Windows and Linux system. Due to the differences in the way new line feeds are handled between the operating systems, the internal checksum algorithm may detect a design file change in this case.

The software reuses the post-synthesis results but re-fits the design if you change the device setting within the same device family. The software reuses the post-fitting netlist if you change only the device speed grade.

Synthesis and Fitter assignments, such as optimization settings, timing assignments, or Fitter location assignments including pin assignments, do not trigger automatic recompilation in the incremental compilation flow. To recompile a partition with new assignments, change the netlist type for that partition to one of the following:

- **Source File** to recompile with all new settings
- **Post-Synthesis** to recompile using existing synthesis results but new Fitter settings
- **Post-Fit** with the **Fitter Preservation Level** set to **Placement** to rerun routing using existing placement results, but new routing settings (such as delay chain settings)

You can use the LogicLock Origin location assignment to change or fine-tune the previous Fitter results from a Post-Fit netlist. For details about how you can affect placement with LogicLock regions, refer to “[Changing Partition Placement with LogicLock Changes](#)” on page 3-50.

Resynthesis Due to Source Code Changes

The Quartus II software uses an internal checksum algorithm to determine whether the contents of a source file have changed. Source files are the design description files used to create the design, and include Memory Initialization Files (.mif) as well as .qxp from exported partitions. When design files in a partition have dependencies on other files, changing one file may initiate an automatic recompilation of another file. The Partition Dependent Files table in the Analysis and Synthesis report lists the design files that contribute to each design partition. You can use this table to determine which partitions are recompiled when a specific file is changed.

For example, if a design has file **A.v** that contains entity **A**, **B.v** that contains entity **B**, and **C.v** that contains entity **C**, then the Partition Dependent Files table for the partition containing entity **A** lists file **A.v**, the table for the partition containing entity **B** lists file **B.v**, and the table for the partition containing entity **C** lists file **C.v**. Any dependencies are transitive, so if file **A.v** depends on **B.v**, and **B.v** depends on **C.v**, the entities in file **A.v** depend on files **B.v** and **C.v**. In this case, files **B.v** and **C.v** are listed in the report table as dependent files for the partition containing entity **A**.



If you use Rapid Recompile, the Quartus II software might not recompile the entire partition from the source code as described in this section; it will reuse compatible results if there have been only small changes to the logic in the partition. Refer to “[Incremental Capabilities Available When A Design Has No Partitions](#)” on page 3-2 for more information.

If you define module parameters in a higher-level module, the Quartus II software checks the parameter values when determining which partitions require resynthesis. If you change a parameter in a higher-level module that affects a lower-level module, the lower-level module is resynthesized. Parameter dependencies are tracked separately from source file dependencies; therefore, parameter definitions are not listed in the **Partition Dependent Files** list.

If a design contains common files, such as an **includes.v** file that is referenced in each entity by the command `'include includes.v`, all partitions are dependent on this file. A change to **includes.v** causes the entire design to be recompiled. The VHDL statement `use work.all` also typically results in unnecessary recompilations, because it makes all entities in the work library visible in the current entity, which results in the current entity being dependent on all other entities in the design.

To avoid this type of problem, ensure that files common to all entities, such as a common include file, contain only the set of information that is truly common to all entities. Remove `use work.all` statements in your VHDL file or replace them by including only the specific design units needed for each entity.

Forcing Use of the Compilation Netlist When a Partition has Changed

Forcing the use of a post-compilation netlist when the contents of a source file has changed is recommended only for advanced users who understand when a partition must be recompiled. You might use this assignment, for example, if you are making source code changes but do not want to recompile the partition until you finish debugging a different partition, or if you are adding simple comments to the source file but you know the design logic itself is not being changed and you want to keep the previous compilation results.

To force the Fitter to use a previously generated netlist even when there are changes to the source files, right-click the partition in the Design Partitions window and then click **Design Partition Properties**. On the **Advanced** tab, turn on the **Ignore changes in source files and strictly use the specified netlist, if available** option.

Turning on this option can result in the generation of a functionally incorrect netlist when source design files change, because source file updates will not be recompiled. Use caution when setting this option.

Exporting Design Partitions from Separate Quartus II Projects

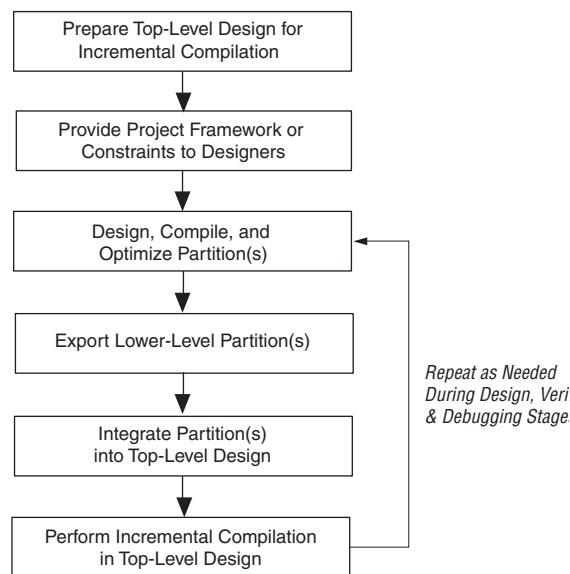
Partitions that are developed by other designers or team members in the same company or third-party IP providers can be exported as design partitions to a Quartus II Exported Partition File (**.qxp**), and then integrated into a top-level design. A **.qxp** is a binary file that contains compilation results describing the exported design partition and includes a post-synthesis netlist, a post-fit netlist, or both, and a set of assignments, sometimes including LogicLock placement constraints. The **.qxp** does not contain the source design files from the original Quartus II project.


To enable team-based development and third-party IP delivery, you can design and optimize partitions in separate copies of the top-level Quartus II project framework, or even in isolation. If the designers have access to the top-level project framework through a source control system, they can access project files as read-only and develop their partition within the source control system. If designers do not have access to a source control system, the project lead can provide the designer with a copy of the top-level project framework to use as they develop their partitions. The project lead also has the option to generate design partition scripts to manage resource and timing budgets in the top-level design when partitions are developed outside the top-level project framework.

The exported compilation results of completed partitions are given to the project lead, preferably using a source control system, who is then responsible for integrating them into the top-level design to obtain a fully functional design. This type of design flow is required only if partition designers want to optimize their placement and routing independently, and pass their design to the project lead to reuse placement and routing results. Otherwise, a project lead can integrate source HDL from several designers in a single Quartus II project, and use the standard incremental compilation flow described previously.

The diagram in Figure 3-7 illustrates the team-based incremental compilation design flow using a methodology in which partitions are compiled in separate Quartus II projects before being integrated into the top-level design. This flow can be used when partitions are developed by other designers or IP providers.

Figure 3-7. Summary of Team-Based Incremental Compilation Flow



 You cannot export or import partitions that have been merged. For more information about merged partitions, refer to “Deciding Which Design Blocks Should Be Design Partitions” on page 3-19.

The topics in this section provide a description of the team-based design flow using exported partitions, describe how to generate a **.qxp** for a design partition, and explain how to integrate the **.qxp** into the top-level design:

There are some additional restrictions related to design flows using exported partitions, described in “Incremental Compilation Restrictions” on page 3-51.

Preparing the Top-Level Design

To prepare your design to incorporate exported partitions, first create the top-level project framework of the design to define the hierarchy for the subdesigns that will be implemented by other team members, designers, or IP providers.

In the top-level design, create project-wide settings, for example, device selection, global assignments for clocks and device I/O ports, and any global signal constraints to specify which signals can use global routing resources.

Next, create the appropriate design partition assignments and set the netlist type for each design partition that will be developed in a separate Quartus II project to **Empty**. Refer to “[Empty Partitions](#)” below for details. It may be necessary to constrain the location of partitions with LogicLock region assignments if they are timing-critical and are expected to change in future compilations, or if the designer or IP provider wants to place and route their design partition independently, to avoid location conflicts. For details, refer to “[Creating a Design Floorplan With LogicLock Regions](#)” on page 3-48.

Finally, provide the top-level project framework to the partition designers, preferably through a source control system. Refer to “[Project Management—Making the Top-Level Design Available to Other Designers](#)” on page 3-32 for more information.

Empty Partitions

You can use a design flow in which some partitions are set to an **Empty** netlist type to develop pieces of the design separately, and then integrate them into the top-level design at a later time. In a team-based design environment, you can set the netlist type to **Empty** for partitions in your design that will be developed by other designers or IP providers. The **Empty** setting directs the Compiler to skip the compilation of a partition and use an empty placeholder netlist for the partition.

When a netlist type is set to **Empty**, peripheral nodes including pins and PLLs are preserved and all other logic is removed. The peripheral nodes including pins help connect the empty partition to the design, and the PLLs help preserve timing of non-empty partitions within empty partitions.

When you set a design partition to **Empty**, a design file is required during Analysis and Synthesis to specify the port interface information so that it can connect the partition correctly to other logic and partitions in the design. If a partition is exported from another project, the **.qxp** contains this information. If there is no **.qxp** or design file to represent the design entity, you must create a wrapper file that defines the design block and specifies the input, output, and bidirectional ports. For example, in Verilog HDL, you should include a module declaration, and in VHDL, you should include an entity and architecture declaration.

Project Management—Making the Top-Level Design Available to Other Designers

In team-based incremental compilation flows, whenever possible, all designers or IP providers should work within the same top-level project framework. Using the same project framework among team members ensures that designers have the settings and constraints needed for their partition, and makes timing closure easier when integrating the partitions into the top-level design. If other designers do not have access to the top-level project framework, the Quartus II software provides tools for passing project information to partition designers.

Distributing the Top-Level Quartus II Project

There are several methods that the project lead can use to distribute the “skeleton” or top-level project framework to other partition designers or IP providers.

- If partition designers have access to the top-level project framework, the project will already include all the settings and constraints needed for the design. This framework should include PLLs and other interface logic if this information is important to optimize partitions.
 - If designers are part of the same design environment, they can check out the required project files from the same source control system. This is the recommended way to share a set of project files.
 - Otherwise, the project lead can provide a copy of the top-level project framework so that each design develops their partition within the same project framework.
- If a partition designer does not have access to the top-level project framework, the project lead can give the partition designer a Tcl script or other documentation to create the separate Quartus II project and all the assignments from the top-level design.

For details about project management scripts you can create with the Quartus II software, refer to [“Optimizing the Placement for a Timing-Critical Partition” on page 3–60](#).

If the partition designers provide the project lead with a post-synthesis .qxp and fitting is performed in the top-level design, integrating the design partitions should be quite easy. If you plan to develop a partition in a separate Quartus II project and integrate the optimized post-fitting results into the top-level design, use the following guidelines to improve the integration process:

- Ensure that a LogicLock region constrains the partition placement and uses only the resources allocated by the project lead.
- Ensure that you know which clocks should be allocated to global routing resources so that there are no resource conflicts in the top-level design.
 - Set the Global Signal assignment to **On** for the high fan-out signals that should be routed on global routing lines.
 - To avoid other signals being placed on global routing lines, turn off **Auto Global Clock and Auto Global Register Controls** under **More Settings** on the Fitter page in the **Settings** dialog box. Alternatively, you can set the Global Signal assignment to **Off** for signals that should not be placed on global routing lines.

Placement for LABs depends on whether the inputs to the logic cells within the LAB use a global clock. You may encounter problems if signals do not use global lines in the partition, but use global routing in the top-level design.

- Use the Virtual Pin assignment to indicate pins of a partition that do not drive pins in the top-level design. This is critical when a partition has more output ports than the number of pins available in the target device. Using virtual pins also helps optimize cross-partition paths for a complete design by enabling you to provide more information about the partition ports, such as location and timing assignments.

- When partitions are compiled independently without any information about each other, you might need to provide more information about the timing paths that may be affected by other partitions in the top-level design. You can apply location assignments for each pin to indicate the port location after incorporation in the top-level design. You can also apply timing assignments to the I/O ports of the partition to perform timing budgeting.



For more information about resource balancing and timing allocation between partitions, refer to the *Best Practices for Incremental Compilation Partitions and Floorplan Assignments* chapter in volume 1 of the *Quartus II Handbook*.

Generating Design Partition Scripts

If IP providers or designers on a team want to optimize their design blocks independently and do not have access to a shared project framework, the project lead must perform some or all of the following tasks to ensure successful integration of the design blocks:

- Determine which assignments should be propagated from the top-level design to the partitions. This requires detailed knowledge of which assignments are required to set up low-level designs.
- Communicate the top-level assignments to the partitions. This requires detailed knowledge of Tcl or other scripting languages to efficiently communicate project constraints.
- Determine appropriate timing and location assignments that help overcome the limitations of team-based design. This requires examination of the logic in the partitions to determine appropriate timing constraints.
- Perform final timing closure and resource conflict avoidance in the top-level design. Because the partitions have no information about each other, meeting constraints at the lower levels does not guarantee they are met when integrated at the top-level. It then becomes the project lead's responsibility to resolve the issues, even though information about the partition implementation may not be available.

Design partition scripts automate the process of transferring the top-level project framework to partition designers in a flow where each design block is developed in separate Quartus II projects before being integrated into the top-level design. If the project lead cannot provide each designer with a copy of the top-level project framework, the Quartus II software provides an interface for managing resources and timing budgets in the top-level design. Design partition scripts make it easier for partition designers to implement the instructions from the project lead, and avoid conflicts between projects when integrating the partitions into the top-level design. This flow also helps to reduce the need to further optimize the designs after integration.

You can use options in the **Generate Design Partition Scripts** dialog box to choose which types of assignments you want to pass down and create in the partitions being developed in separate Quartus II projects.

For an example design scenario using design partition scripts, refer to “[Enabling Designers on a Team to Optimize Independently](#)” on page 3-43.

- ❓ For step-by-step information on how to generate design partition scripts, and a description of each option that can be included in design partition scripts, refer to *Generating Design Partition Scripts for Project Management*, and *Generate Design Partition Scripts Dialog Box* in Quartus II Help.

Exporting Partitions

When partition designers achieve the design requirements in their separate Quartus II projects, each designer can export their design as a partition so it can be integrated into the top-level design by the project lead. The **Export Design Partition** dialog box, available from the Project menu, allows designers to export a design partition to a Quartus II Exported Partition File (.qxp) with a post-synthesis netlist, a post-fit netlist, or both. The project lead then adds the .qxp to the top-level design to integrate the partition.

A designer developing a timing-critical partition or who wants to optimize their partition on their own would opt to export their completed partition with a post-fit netlist, allowing for the partition to more reliably meet timing requirements after integration. In this case, you must ensure that resources are allocated appropriately to avoid conflicts. If the placement and routing optimization can be performed in the top-level design, exporting a post-synthesis netlist allows the most flexibility in the top-level design and avoids potential placement or routing conflicts with other partitions.

When designing the partition logic to be exported into another project, you can add logic around the design block to be exported as a design partition. You can instantiate additional design components for the Quartus II project so that it matches the top-level design environment, especially in cases where you do not have access to the full top-level design project. For example, you can include a top-level PLL in the project, outside of the partition to be exported, so that you can optimize the design with information about the frequency multipliers, phase shifts, compensation delays, and any other PLL parameters. The software then captures timing and resource requirements more accurately while ensuring that the timing analysis in the partition is complete and accurate. You can export the partition for the top-level design without any auxiliary components that are instantiated outside the partition being exported.

If your design team uses makefiles and design partition scripts, the project lead can use the **make** command with the **master_makefile** command created by the scripts to export the partitions and create .qxp files. When a partition has been compiled and is ready to be integrated into the top-level design, you can export the partition with option on the **Export Design Partition** dialog box, available from the Project menu.

- ❓ For more information about how to export a design partition, refer to *Using a Team-Based Incremental Compilation Design Flow* in the Quartus II Help.

Viewing the Contents of a Quartus II Exported Partition File (.qxp)

The QXP report allows you to view a summary of the contents in a .qxp when you open the file in the Quartus II software. The .qxp is a binary file that contains compilation results so the file cannot be read in a text editor. The QXP report opens in the main Quartus II window and contains summary information including a list of the I/O ports, resource usage summary, and a list of the assignments used for the exported partition.

Integrating Partitions into the Top-Level Design

To integrate a partition developed in a separate Quartus II project into the top-level design, you can simply add the **.qxp** as a source file in your top-level design (just like a Verilog or VHDL source file). You can also use the **Import Design Partition** dialog box to import the partition, in certain situations, described in “[Advanced Importing Options](#)” on page 3-37.

The **.qxp** contains the design block exported from the partition and has the same name as the partition. When you instantiate the design block into a top-level design and include the **.qxp** as a source file, the software adds the exported netlist to the database for the top-level design. The **.qxp** port names are case sensitive if the original HDL of the partition was case sensitive.

When you use a **.qxp** as a source file in this way, you can choose whether you want the **.qxp** to be a partition in the top-level design. If you do not designate the **.qxp** instance as a partition, the software reuses just the post-synthesis compilation results from the **.qxp**, removes unconnected ports and unused logic just like a regular source file, and then performs placement and routing.

If you assigned the **.qxp** instance as a partition, you can set the netlist type in the Design Partitions Window to choose the level of results to preserve from the **.qxp**. To preserve the placement and routing results from the exported partition, set the netlist type to **Post-Fit** for the **.qxp** partition in the top-level design. If you assign the instance as a design partition, the partition boundary is preserved, as discussed in “[Impact of Design Partitions on Design Optimization](#)” on page 3-20.

Integrating Assignments from the .qxp

The Quartus II software filters assignments from **.qxp** files to include appropriate assignments in the top-level design. The assignments in the **.qxp** are treated like assignments made in an HDL source file, and are not listed in the Quartus II Settings File (**.qsf**) for the top-level design. Most assignments from the **.qxp** can be overridden by assignments in the top-level design.

The following subsections provide more details about specific assignment types:

Design Partition Assignments Within the Exported Partition

Design partition assignments defined within a separate Quartus II project are not added to the top-level design. All logic under the exported partition in the project hierarchy is treated as single instance in the **.qxp**.

Synopsys Design Constraint Files for the Quartus II TimeQuest Timing Analyzer

Timing assignments made for the Quartus II TimeQuest analyzer in a Synopsys Design Constraint File (**.sdc**) in the lower-level partition project are not added to the top-level design. Ensure that the top-level design includes all of the timing requirements for the entire project.



For recommendations about managing SDC constraints for the top-level design and independent lower-level partition projects, refer to the *Best Practices for Incremental Compilation Partitions and Floorplan Assignments* chapter in volume 1 of the *Quartus II Handbook*.

Global Assignments

The project lead should make all global project-wide assignments in the top-level design. Global assignments from the exported partition's project are not added to the top-level design. When it is possible for a particular constraint, the global assignment is converted to an instance-specific assignment for the exported design partition.

LogicLock Region Assignments

The project lead typically creates LogicLock region assignments in the top-level design for any lower-level partition designs where designer or IP providers plan to export post-fit information to be used in the top-level design, to help avoid placement conflicts between partitions. When you use the **.qxp** as a source file, LogicLock constraints from the exported partition are applied in the top-level design, but will not appear in your **.qsf** file or LogicLock Regions window for you to view or edit. The LogicLock region itself is not required to constrain the partition placement in the top-level design if the netlist type is set to **Post-Fit**, because the netlist contains all the placement information. For information on how to control LogicLock region assignments for exported partitions, refer to the [“Advanced Importing Options”](#) on page 3-37.

Integrating Encrypted IP Cores from .qxp Files

Proper license information is required to compile encrypted IP cores. If an IP core is exported as a **.qxp** from another Quartus II project, the top-level designer instantiating the **.qxp** must have the correct license. The software requires a full license to generate an unrestricted programming file. If you do not have a license, but the IP in the **.qxp** was compiled with OpenCore Plus hardware evaluation support, you can generate an evaluation programming file without a license. If the IP supports OpenCore simulation only, you can fully compile the design and generate a simulation netlist, but you cannot create programming files unless you have a full license.

Advanced Importing Options

You can use advanced options in the **Import Design Partition** dialog box to integrate a partition developed in a separate Quartus II project into the top-level design. The import process adds more control than using the **.qxp** as a source file, and is useful only in the following circumstances:

- **If you want LogicLock regions in your top-level design (.qsf)**—If you have regions in your partitions that are not also in the top-level design, the regions will be added to your **.qsf** during the import process.
- **If you want different settings or placement for different instantiations of the same entity**—You can control the setting import process with the advanced import options, and specify different settings for different instances of the same **.qxp** design block.

When you use the **Import Design Partition** dialog box to integrate a partition into the top-level design, the import process sets the partition's netlist type to **Imported** in the Design Partitions window.

After you compile the entire design, if you make changes to the place-and-route results (such as movement of an imported LogicLock region), use the **Post-Fit** netlist type on subsequent compilations. To discard an imported netlist and recompile from source code, you can compile the partition with the netlist type set to **Source File** and be sure to include the relevant source code in the top-level design. Refer to “[Netlist Type for Design Partitions](#)” on page 3-25 for details. The import process sets the partition’s Fitter Preservation Level to the setting with the highest degree of preservation supported by the imported netlist. For example, if a post-fit netlist is imported with placement information, the Fitter Preservation Level is set to **Placement**, but you can change it to the **Netlist Only** value. For more information about preserving previous compilation results, refer to “[Netlist Type for Design Partitions](#)” on page 3-25 and “[Fitter Preservation Level for Design Partitions](#)” on page 3-26.

When you import a partition from a **.qxp**, the **.qxp** itself is not part of the top-level design because the netlists from the file have been imported into the project database. Therefore if a new version of a **.qxp** is exported, the top-level designer must perform another import of the **.qxp**.

When you import a partition into a top-level design with the **Import Design Partition** dialog box, the software imports relevant assignments from the partition into the top-level design, as described for the source file integration flow in “[Integrating Assignments from the .qxp](#)” on page 3-36. If required, you can change the way some assignments are imported, as described in the following subsections.

Importing LogicLock Assignments

LogicLock regions are set to a fixed size when imported. If you instantiate multiple instances of a subdesign in the top-level design, the imported LogicLock regions are set to a Floating location. Otherwise, they are set to a Fixed location. You can change the location of LogicLock regions after they are imported, or change them to a Floating location to allow the software to place each region but keep the relative locations of nodes within the region wherever possible. For details, refer to “[Changing Partition Placement with LogicLock Changes](#)” on page 3-50. To preserve changes made to a partition after compilation, use the **Post-Fit** netlist type.

The LogicLock Member State assignment is set to **Locked** to signify that it is a preserved region.

LogicLock back-annotation and node location data is not imported because the **.qxp** contains all of the relevant placement information. Altera strongly recommends that you do not add to or delete members from an imported LogicLock region.

Advanced Import Settings

The **Advanced Import Settings** dialog box allows you to disable assignment import and specify additional options that control how assignments and regions are integrated when importing a partition into a top-level design, including how to resolve assignment conflicts.

- ② For descriptions of the advanced import options available, refer to *Advanced Import Settings Dialog Box* in Quartus II Help.

Team-Based Design Optimization and Third-Party IP Delivery Scenarios

This section includes the following design flows with step-by-step descriptions when you have partitions being developed in separate Quartus II projects, or by a third-party IP provider.

- “Using an Exported Partition to Send to a Design Without Including Source Files” on page 3-39
- “Creating Precompiled Design Blocks (or Hard-Wired Macros) for Reuse” on page 3-40
- “Designing in a Team-Based Environment” on page 3-42
- “Enabling Designers on a Team to Optimize Independently” on page 3-43
- “Performing Design Iterations With Lower-Level Partitions” on page 3-47

Using an Exported Partition to Send to a Design Without Including Source Files

Scenario background: A designer wants to produce a design block and needs to send out their design, but to preserve their IP, they prefer to send a synthesized netlist instead of providing the HDL source code to the recipient. You can use this flow to implement a black box.

Use this flow to package a full design as a single source file to send to an end customer or another design location.

As the sender in this scenario perform the following steps to export a design block:

1. Provide the device family name to the recipient. If you send placement information with the synthesized netlist, also provide the exact device selection so they can set up their project to match.
2. Create a black box wrapper file that defines the port interface for the design block and provide it to the recipient for instantiating the block as an empty partition in the top-level design.
3. Create a Quartus II project for the design block, and complete the design.
4. Export the level of hierarchy into a single **.qxp**. Following a successful compilation of the project, you can generate a **.qxp** from the GUI, the command-line, or with Tcl commands, as described in the following:
 - If you are using the Quartus II GUI, use the **Export Design Partition** dialog box.
 - If you are using command-line executables, run **quartus_cdb** with the **--incremental_compilation_export** option.
 - If you are using Tcl commands, use the following command:
`execute_flow -incremental_compilation_export.`
5. Select the option to include just the **Post-synthesis netlist** if you do not have to send placement information. If the recipient wants to reproduce your exact Fitter results, you can select the **Post-fitting netlist** option, and optionally enable **Export routing**.

6. If a partition contains sub-partitions, then the sub-partitions are automatically flattened and merged into the partition netlist before exporting. You can change this behavior and preserve the sub-partition hierarchy by turning off the **Flatten sub-partitions** option on the **Export Design Partition** dialog box. Optionally, you can use the `-dont_flatten` sub-option for the `export_partition` Tcl command.
7. Provide the **.qxp** to the recipient. Note that you do not have to send any of your design source code.

As the recipient in this example, first create a Quartus II project for your top-level design and ensure that your project targets the same device (or at least the same device family if the **.qxp** does not include placement information), as specified by the IP designer sending the design block. Instantiate the design block using the port information provided, and then incorporate the design block into a top-level design.

Add the **.qxp** from the IP designer as a source file in your Quartus II project to replace any empty wrapper file. If you want to use just the post-synthesis information, you can choose whether you want the file to be a partition in the top-level design. To use the post-fit information from the **.qxp**, assign the instance as a design partition and set the netlist type to **Post-Fit**. Refer to “[Creating Design Partitions](#)” on page 3-9 and “[Netlist Type for Design Partitions](#)” on page 3-25.

Creating Precompiled Design Blocks (or Hard-Wired Macros) for Reuse

Scenario background: An IP provider wants to produce and sell an IP core for a component to be used in higher-level systems. The IP provider wants to optimize the placement of their block for maximum performance in a specific Altera device and then deliver the placement information to their end customer. To preserve their IP, they also prefer to send a compiled netlist instead of providing the HDL source code to their customer.

Use this design flow to create a precompiled IP block (sometimes known as a hard-wired macro) that can be instantiated in a top-level design. This flow provides the ability to export a design block with post-synthesis or placement (and, optionally, routing) information and to import any number of copies of this pre-compiled block into another design.

The customer first specifies which Altera device is being used for this project and provides the design specifications.

As the IP provider in this example, perform the following steps to export a preplaced IP core (or hard macro):

1. Create a black box wrapper file that defines the port interface for the IP core and provide the file to the customer to instantiate as an empty partition in the top-level design.
2. Create a Quartus II project for the IP core.
3. Create a LogicLock region for the design hierarchy to be exported.



Using a LogicLock region for the IP core allows the customer to create an empty placeholder region to reserve space for the IP in the design floorplan and ensures that there are no conflicts with the top-level design logic. Reserved space also helps ensure the IP core does not affect the timing performance of other logic in the top-level design. Additionally, with a

LogicLock region, you can preserve placement either absolutely or relative to the origin of the associated region. This is important when a **.qxp** is imported for multiple partition hierarchies in the same project, because in this case, the location of at least one instance in the top-level design does not match the location used by the IP provider.

4. If required, add any logic (such as PLLs or other logic defined in the customer's top-level design) around the design hierarchy to be exported. If you do so, create a design partition for the design hierarchy that will be exported as an IP core.
5. Optimize the design and close timing to meet the design specifications.
6. Export the level of hierarchy for the IP core into a single **.qxp**.
7. Provide the **.qxp** to the customer. Note that you do not have to send any of your design source code to the customer; the design netlist and placement and routing information is contained within the **.qxp**.

As the customer in this example, incorporate the IP core in your design by performing the following steps:

1. Create a Quartus II project for the top-level design that targets the same device and instantiate a copy or multiple copies of the IP core. Use a black box wrapper file to define the port interface of the IP core.
2. Perform Analysis and Elaboration to identify the design hierarchy.
3. Create a design partition for each instance of the IP core (refer to “[Creating Design Partitions](#)” on page 3-57) with the netlist type set to **Empty** (refer to “[Netlist Type for Design Partitions](#)” on page 3-25).
4. You can now continue work on your part of the design and accept the IP core from the IP provider when it is ready.
5. Include the **.qxp** from the IP provider in your project to replace the empty wrapper-file for the IP instance. Or, if you are importing multiple copies of the design block and want to import relative placement, follow these additional steps:
 - a. Use the **Import** command to select each appropriate partition hierarchy. You can import a **.qxp** from the GUI, the command-line, or with Tcl commands:
 - If you are using the Quartus II GUI, use the **Import Design Partition** command.
 - If you are using command-line executables, run **quartus_cdb** with the **--incremental_compilation_import** option.
 - If you are using Tcl commands, use the following command:
`execute_flow -incremental_compilation_import.`
 - b. When you have multiple instances of the IP block, you can set the imported LogicLock regions to floating, or move them to a new location, and the software preserves the relative placement for each of the imported modules (relative to the origin of the LogicLock region). Routing information is preserved whenever possible. Refer to “[Changing Partition Placement with LogicLock Changes](#)” on page 3-50



The Fitter ignores relative placement assignments if the LogicLock region's location in the top-level design is not compatible with the locations exported in the .qxp.

6. You can control the level of results preservation with the **Netlist Type** setting. Refer to "Netlist Type for Design Partitions" on page 3-25.



If the IP provider did not define a LogicLock region in the exported partition, the software preserves absolute placement locations and this leads to placement conflicts if the partition is imported for more than one instance.

Designing in a Team-Based Environment

Scenario background: A project includes several lower-level design blocks that are developed separately by different designers and instantiated exactly once in the top-level design.

This scenario describes how to use incremental compilation in a team-based design environment where each designer has access to the top-level project framework, but wants to optimize their design in a separate Quartus II project before integrating their design block into the top-level design.

As the project lead in this scenario, perform the following steps to prepare the top-level design:

1. Create a new Quartus II project to ultimately contain the full implementation of the entire design and include a "skeleton" or framework of the design that defines the hierarchy for the subdesigns implemented by separate designers. The top-level design implements the top-level entity in the design and instantiates wrapper files that represent each subdesign by defining only the port interfaces but not the implementation.
2. Make project-wide settings. Select the device, make global assignments such as device I/O ports, define the top-level timing constraints, and make any global signal allocation constraints to specify which signals can use global routing resources.
3. Make design partition assignments for each subdesign and set the netlist type for each design partition to be imported to **Empty** in the Design Partitions window.
4. Create LogicLock regions to create a design floorplan for each of the partitions that will be developed separately. This floorplan should consider the connectivity between partitions and estimates of the size of each partition based on any initial implementation numbers and knowledge of the design specifications.

5. Provide the top-level project framework to partition designers using one of the following procedures:
 - Allow access to the full project for all designers through a source control system. Each designer can check out the projects files as read-only and work on their blocks independently. This design flow provides each designer with the most information about the full design, which helps avoid resource conflicts and makes design integration easy.
 - Provide a copy of the top-level Quartus II project framework for each designer. You can use the **Copy Project** command on the Project menu or create a project archive.

As the designer of a lower-level design block in this scenario, design and optimize your partition in your copy of the top-level design, and then follow these steps when you have achieved the desired compilation results:

1. On the Project menu, click **Export Design Partition**.
2. In the **Export Design Partition** dialog box, choose the netlist(s) to export. You can export a Post-synthesis netlist if placement or performance preservation is not required, to provide the most flexibility for the Fitter in the top-level design. Select Post-fit netlist to preserve the placement and performance of the lower-level design block, and turn on **Export routing** to include the routing information, if required. One **.qxp** can include both post-synthesis and post-fitting netlists.
3. Provide the **.qxp** to the project lead.

Finally, as the project lead in this scenario, perform these steps to integrate the **.qxp** files received from designers of each partition:

1. Add the **.qxp** as a source file in the Quartus II project, to replace any empty wrapper file for the previously **Empty** partition.
2. Change the netlist type for the partition from **Empty** to the required level of results preservation.

Enabling Designers on a Team to Optimize Independently

Scenario background: A project consists of several lower-level design blocks that are developed separately by different designers who do not have access to a shared top-level project framework. This scenario is similar to “[Creating Precompiled Design Blocks \(or Hard-Wired Macros\) for Reuse](#)” on page 3-40, but assumes that there are several design blocks being developed independently (instead of just one IP block), and the project lead can provide some information about the design to the individual designers. If the designers have shared access to the top-level design, use the previous scenario “[Designing in a Team-Based Environment](#)” on page 3-42.

This scenario describes how to use incremental compilation in a team-based design environment where designers or IP developers want to fully optimize the placement and routing of their design independently in a separate Quartus II project before sending the design to the project lead. This design flow requires more planning and careful resource allocation because design blocks are developed independently.

As the project lead in this scenario, perform the following steps to prepare the top-level design:

1. Create a new Quartus II project to ultimately contain the full implementation of the entire design and include a “skeleton” or framework of the design that defines the hierarchy for the subdesigns implemented by separate designers. The top-level design implements the top-level entity in the design and instantiates wrapper files that represent each subdesign by defining only the port interfaces but not the implementation.
2. Make project-wide settings. Select the device, make global assignments such as device I/O ports, define the top-level timing constraints, and make any global signal constraints to specify which signals can use global routing resources.
3. Make design partition assignments for each subdesign and set the netlist type for each design partition to be imported to **Empty** in the Design Partitions window.
4. Create LogicLock regions. This floorplan should consider the connectivity between partitions and estimates of the size of each partition based on any initial implementation numbers and knowledge of the design specifications.
5. Provide the constraints from the top-level design to partition designers using one of the following procedures:
 - Use design partition scripts to pass constraints and generate separate Quartus II projects. On the Project menu, use the **Generate Design Partition Scripts** command, or run the script generator from a Tcl or command prompt. Make changes to the default script options as required for your project. Altera recommends that you pass all the default constraints, including LogicLock regions, for all partitions and virtual pin location assignments. If partitions have not already been created by the other designers, use the partition script to set up the projects so that you can easily take advantage of makefiles. Provide each partition designer with the Tcl file to create their project with the appropriate constraints. If you are using makefiles, provide the makefile for each partition.
 - Use documentation or manually-created scripts to pass all constraints and assignments to each partition designer.

As the designer of a lower-level design block in this scenario, perform the appropriate set of steps to successfully export your design, whether the design team is using makefiles or exporting and importing the design manually.

If you are using makefiles with the design partition scripts, perform the following steps:

1. Use the **make** command and the makefile provided by the project lead to create a Quartus II project with all design constraints, and compile the project.
2. The information about which source file should be associated with which partition is not available to the software automatically, so you must specify this information in the makefile. You must specify the dependencies before the software rebuilds the project after the initial call to the makefile.
3. When you have achieved the desired compilation results and the design is ready to be imported into the top-level design, the project lead can use the `master_makefile` command to export this partition and create a **.qxp**, and then import it into the top-level design.

If you are not using makefiles, perform the following steps:

1. If you are using design partition scripts, source the Tcl script provided by the Project Lead to create a project with the required settings:
 - To source the Tcl script in the Quartus II software, on the Tools menu, click **Utility Windows** to open the Tcl console. Navigate to the script's directory, and type the following command: `source <filename>` ↵
 - To source the Tcl script at the system command prompt, type the following command: `quartus_cdb -t <filename>.tcl` ↵
2. If you are not using design partition scripts, create a new Quartus II project for the subdesign, and then apply the following settings and constraints to ensure successful integration:
 - Make LogicLock region assignments and global assignments (including clock settings) as specified by the project lead.
 - Make Virtual Pin assignments for ports which represent connections to core logic instead of external device pins in the top-level design.
 - Make floorplan location assignments to the Virtual Pins so they are placed in their corresponding regions as determined by the top-level design. This provides the Fitter with more information about the timing constraints between modules. Alternatively, you can apply timing I/O constraints to the paths that connect to virtual pins.
3. Proceed to compile and optimize the design as needed.
4. When you have achieved the desired compilation results, on the Project menu, click **Export Design Partition**.
5. In the **Export Design Partition** dialog box, choose the netlist(s) to export. You can export a Post-synthesis netlist instead if placement or performance preservation is not required, to provide the most flexibility for the Fitter in the top-level design. Select **Post-fit** to preserve the placement and performance of the lower-level design block, and turn on Export routing to include the routing information, if required. One **.qxp** can include both post-synthesis and post-fitting netlists.
6. Provide the **.qxp** to the project lead.

Finally, as the project lead in this scenario, perform the appropriate set of steps to import the **.qxp** files received from designers of each partition.

If you are using makefiles with the design partition scripts, perform the following steps:

1. Use the `master_makefile` command to export each partition and create **.qxp** files, and then import them into the top-level design.
2. The software does not have all the information about which source files should be associated with which partition, so you must specify this information in the makefile. The software cannot rebuild the project if source files change unless you specify the dependencies.

If you are not using makefiles, perform the following steps:

1. Add the **.qxp** as a source file in the Quartus II project, to replace any empty wrapper file for the previously Empty partition.

2. Change the netlist type for the partition from Empty to the required level of results preservation.

Resolving Assignment Conflicts During Integration

When integrating lower-level design blocks, the project lead may notice some assignment conflicts. This can occur, for example, if the lower-level design block designers changed their LogicLock regions to account for additional logic or placement constraints, or if the designers applied I/O port timing constraints that differ from constraints added to the top-level design by the project lead. The project lead can address these conflicts by explicitly importing the partitions into the top-level design, and using options in the **Advanced Import Settings** dialog box, as described in “[Advanced Importing Options](#)” on page 3-37. After the project lead obtains the **.qxp** for each lower-level design block from the other designers, use the **Import Design Partition** command on the Project menu and specify the partition in the top-level design that is represented by the lower-level design block **.qxp**. Repeat this import process for each partition in the design. After you have imported each partition once, you can select all the design partitions and use the **Reimport using latest import files at previous locations** option to import all the files from their previous locations at one time. To address assignment conflicts, the project lead can take one or both of the following actions:

- Allow new assignments to be imported
- Allow existing assignments to be replaced or updated

When LogicLock region assignment conflicts occur, the project lead may take one of the following actions:

- Allow the imported region to replace the existing region
- Allow the imported region to update the existing region
- Skip assignment import for regions with conflicts

If the placement of different lower-level design blocks conflict, the project lead can also set the partition's **Fitter Preservation Level** to **Netlist Only**, which allows the software to re-perform placement and routing with the imported netlist.

Importing a Partition to be Instantiated Multiple Times

In this variation of the design scenario, one of the lower-level design blocks is instantiated more than once in the top-level design. The designer of the lower-level design block may want to compile and optimize the entity once under a partition, and then import the results as multiple partitions in the top-level design.

If you import multiple instances of a lower-level design block into the top-level design, the imported LogicLock regions are automatically set to Floating status.

If you resolve conflicts manually, you can use the import options and manual LogicLock assignments to specify the placement of each instance in the top-level design.

Performing Design Iterations With Lower-Level Partitions

Scenario background: A project consists of several lower-level subdesigns that have been exported from separate Quartus II projects and imported into the top-level design. In this example, integration at the top level has failed because the timing requirements are not met. The timing requirements might have been met in each individual lower-level project, but critical inter-partition paths in the top-level design are causing timing requirements to fail.

After trying various optimizations in the top-level design, the project lead determines that the design cannot meet the timing requirements given the current partition placements that were imported. The project lead decides to pass additional information to the lower-level partitions to improve the placement.

Use this flow if you re-optimize partitions exported from separate Quartus II projects by incorporating additional constraints from the integrated top-level design.

The best way to provide top-level design information to designers of lower-level partitions is to provide the complete top-level project framework using the following steps:

1. For all partitions other than the one(s) being optimized by a designer(s) in a separate Quartus II project(s), set the netlist type to **Post-Fit**.
2. Make the top-level design directory available in a shared source control system, if possible. Otherwise, copy the entire top-level design project directory (including database files), or create a project archive including the post-compilation database.
3. Provide each partition designer with a checked-out version or copy of the top-level design.
4. The partition designers recompile their designs within the new project framework that includes the rest of the design's placement and routing information as well top-level resource allocations and assignments, and optimize as needed.
5. When the results are satisfactory and the timing requirements are met, export the updated partition as a **.qxp**.

If this design flow is not possible, you can generate partition-specific scripts for individual designs to provide information about the top-level project framework with these steps:

1. In the top-level design, on the Project menu, click **Generate Design Partition Scripts**, or launch the script generator from Tcl or the command line.
2. If lower-level projects have already been created for each partition, you can turn off the **Create lower-level project if one does not exist** option.
3. Make additional changes to the default script options, as necessary. Altera recommends that you pass all the default constraints, including LogicLock regions, for all partitions and virtual pin location assignments. Altera also recommends that you add a maximum delay timing constraint for the virtual I/O connections in each partition.

4. The Quartus II software generates Tcl scripts for all partitions, but in this scenario, you would focus on the partitions that make up the cross-partition critical paths. The following assignments are important in the script:
 - Virtual pin assignments for module pins not connected to device I/O ports in the top-level design.
 - Location constraints for the virtual pins that reflect the initial top-level placement of the pin's source or destination. These help make the lower-level placement "aware" of its surroundings in the top-level design, leading to a greater chance of timing closure during integration at the top level.
 - INPUT_MAX_DELAY and OUTPUT_MAX_DELAY timing constraints on the paths to and from the I/O pins of the partition. These constrain the pins to optimize the timing paths to and from the pins.
5. The partition designers source the file provided by the project lead.
 - To source the Tcl script from the Quartus II GUI, on the Tools menu, click **Utility Windows** and open the Tcl console. Navigate to the script's directory, and type the following command: `source <filename>` ←
 - To source the Tcl script at the system command prompt, type the following command: `quartus_cdb -t <filename>.tcl` ←
6. The partition designers recompile their designs with the new project information or assignments and optimize as needed. When the results are satisfactory and the timing requirements are met, export the updated partition as a **.qxp**.

The project lead obtains the updated **.qxp** files from the partition designers and adds them to the top-level design. When a new **.qxp** is added to the files list, the software will detect the change in the "source file" and use the new **.qxp** results during the next compilation. If the project uses the advanced import flow, the project lead must perform another import of the new **.qxp**.

You can now analyze the design to determine whether the timing requirements have been achieved. Because the partitions were compiled with more information about connectivity at the top level, it is more likely that the inter-partition paths have improved placement which helps to meet the timing requirements.

Creating a Design Floorplan With LogicLock Regions

A floorplan represents the layout of the physical resources on the device. Creating a design floorplan, or floorplanning, describe the process of mapping the logical design hierarchy onto physical regions in the device floorplan. After you have partitioned the design, you can create floorplan location assignments for the design to improve the quality of results when using the incremental compilation design flow. Creating a design floorplan is not a requirement to use an incremental compilation flow, but it is recommended in certain cases. Floorplan location planning can be important for a design that uses incremental compilation for the following reasons:

- To avoid resource conflicts between partitions, predominantly when partitions are imported from another Quartus II project
- To ensure a good quality of results when recompiling individual timing-critical partitions

Design floorplan assignments prevent the situation in which the Fitter must place a partition in an area of the device where most resources are already used by other partitions. A physical region assignment provides a reasonable region to re-place logic after a change, so the Fitter does not have to scatter logic throughout the available space in the device.

Floorplan assignments are not required for non-critical partitions compiled as part of the top-level design. The logic for partitions that are not timing-critical (such as simple top-level glue logic) can be placed anywhere in the device on each recompilation, if that is best for your design.

The simplest way to create a floorplan for a partitioned design is to create one LogicLock region per partition (including the top-level partition). If you have a compilation result for a partitioned design with no LogicLock regions, you can use the Chip Planner with the Design Partition Planner to view the partition placement in the device floorplan. You can draw regions in the floorplan that match the general location and size of the logic in each partition. Or, initially, you can set each region with the default settings of **Auto** size and **Floating** location to allow the Quartus II software to determine the preliminary size and location for the regions. Then, after compilation, use the Fitter-determined size and origin location as a starting point for your design floorplan. Check the quality of results obtained for your floorplan location assignments and make changes to the regions as needed. Alternatively, you can perform synthesis, and then set the regions to the required size based on resource estimates. In this case, use your knowledge of the connections between partitions to place the regions in the floorplan.

Once you have created an initial floorplan, you can refine the region using tools in the Quartus II software. You can also use advanced techniques such as creating non-rectangular regions by merging LogicLock regions.



For more information about when creating a design floorplan can be important, as well as guidelines for creating the floorplan, refer to the *Best Practices for Incremental Compilation Partitions and Floorplan Assignments* chapter in volume 1 of the *Quartus II Handbook*.

You can use the Incremental Compilation Advisor to check that your LogicLock regions meet Altera's guidelines, as described in "[Incremental Compilation Advisor](#)" on page 3-24.

Creating and Manipulating LogicLock Regions

Options in the **LogicLock Regions Properties** dialog box, available from the Assignments menu, allow you to enter specific sizing and location requirements for a region. You can also view and refine the size and location of LogicLock regions in the Quartus II Chip Planner. You can select a region in the graphical interface in the Chip Planner and use handles to move or resize the region.

Options in the **Layer Settings** panel in the Chip Planner allow you to create, delete, and modify tasks to determine which objects, including LogicLock regions and design partitions, to display in the Chip Planner.

- ❓ For more information about creating and viewing LogicLock regions in the LogicLock Regions window and Chip Planner, refer to *Creating and Manipulating LogicLock Regions* in Quartus II Help.

Changing Partition Placement with LogicLock Changes

When a partition is assigned to a LogicLock region as part of a design floorplan, you can modify the placement of a post-fit partition by moving the LogicLock region. As described in “What Changes Initiate the Automatic Resynthesis of a Partition?” on page 3-28, most assignment changes do not initiate a recompilation of a partition if the netlist type specifies that Fitter results should be preserved. For example, changing a pin assignment does not initiate a recompilation; therefore, the design does not use the new pin assignment unless you change the netlist type to **Post-Synthesis** or **Source File**.

Similarly, if a partition's placement is preserved, and the partition is assigned to a LogicLock region, the Fitter always reuses the corresponding LogicLock region size specified in the post-fit netlist. That is, changes to the LogicLock **Size** setting do not initiate refitting if a partition's placement is preserved with the **Post-Fit** netlist type, or with **.qxp** that includes post-fit information.

However, you can use the LogicLock **Origin** location assignment to change or fine-tune the previous Fitter results. When you change the **Origin** setting for a region, the Fitter can move the region in the following manner, depending upon how the placement is preserved for that region's members:

- When you set a new region Origin, the Fitter uses the new origin and replaces the logic, preserving the relative placement of the member logic.
- When you set the region Origin to **Floating**, the following conditions apply:
 - If the region's member placement is preserved with an imported partition, the Fitter chooses a new Origin and re-places the logic, preserving the relative placement of the member logic within the region.
 - If the region's member placement is preserved with a **Post-Fit** netlist type, the Fitter does not change the Origin location, and reuses the previous placement results.

Taking Advantage of the Early Timing Estimator

When creating a floorplan you can take advantage of the Early Timing Estimator to enable quick compilations of the design while creating assignments. The Early Timing Estimator feature provides a timing estimate for a design without having to run a full compilation. You can use the Chip Planner to view the “placement estimate” created by this feature, identify critical paths by locating from the timing analyzer reports, and, if necessary, add or modify floorplan constraints. You can then rerun the Early Timing Estimator to quickly assess the impact of any floorplan location assignments or logic changes, enabling rapid iterations on design variants to help you find the best solution. This faster placement has an impact on the quality of results. If getting the best quality of results is important in a given design iteration, perform a full compilation with the Fitter instead of using the Early Timing Estimate feature.

Incremental Compilation Restrictions

This section documents the following restrictions and limitations that you may encounter when using incremental compilation, including interactions with other Quartus II features:

- “When Timing Performance May Not Be Preserved Exactly” on page 3-51
- “When Placement and Routing May Not Be Preserved Exactly” on page 3-51
- “Using Incremental Compilation With Quartus II Archive Files” on page 3-52
- “Formal Verification Support” on page 3-52
- “SignalProbe Pins and Engineering Change Orders” on page 3-52
- “SignalTap II Logic Analyzer in Exported Partitions” on page 3-53
- “External Logic Analyzer Interface in Exported Partitions” on page 3-53
- “Assignments Made in HDL Source Code in Exported Partitions” on page 3-54
- “Design Partition Script Limitations” on page 3-54
- “Restrictions on Megafunction Partitions” on page 3-56
- “Register Packing and Partition Boundaries” on page 3-56
- “I/O Register Packing” on page 3-56

When Timing Performance May Not Be Preserved Exactly

Timing performance might change slightly in a partition with placement and routing preserved when other partitions are incorporated or re-placed and routed. Timing changes are due to changes in parasitic loading or crosstalk introduced by the other (changed) partitions. These timing changes are very small, typically less than 30 ps on a timing path. Additional fan-out on routing lines when partitions are added can also degrade timing performance.

To ensure that a partition continues to meet its timing requirements when other partitions change, a very small timing margin might be required. The Fitter automatically works to achieve such margin when compiling any design, so you do not need to take any action.

When Placement and Routing May Not Be Preserved Exactly

The Fitter may have to refit affected nodes if the two nodes are assigned to the same location, due to imported netlists or empty partitions set to re-use a previous post-fit netlist. There are two cases in which routing information cannot be preserved exactly. First, when multiple partitions are imported, there might be routing conflicts because two lower-level blocks could be using the same routing wire, even if the floorplan assignments of the lower-level blocks do not overlap. These routing conflicts are automatically resolved by the Quartus II Fitter re-routing on the affected nets. Second, if an imported LogicLock region is moved in the top-level design, the relative placement of the nodes is preserved but the routing cannot be preserved, because the routing connectivity is not perfectly uniform throughout a device.

Using Incremental Compilation With Quartus II Archive Files

The post-synthesis and post-fitting netlist information for each design partition is stored in the project database, the **incremental_db** directory. When you archive a project, the database information is not included in the archive unless you include the compilation database in the **.qar** file.

If you want to re-use post-synthesis or post-fitting results, include the database files in the **Archive Project** dialog box so compilation results are preserved. Click **Advanced**, and choose a file set that includes the compilation database, or turn on **Incremental compilation database files** to create a Custom file set.

When you include the database, the file size of the **.qar** archive file may be significantly larger than an archive without the database.

The netlist information for imported partitions is already saved in the corresponding **.qxp**. Imported **.qxp** files are automatically saved in a subdirectory called **imported_partitions**, so you do not need to archive the project database to keep the results for imported partitions. When you restore a project archive, the partition is automatically reimported from the **.qxp** in this directory if it is available.

For new device families with advanced support, a version-compatible database might not be available. In this case, the archive will not include the compilation database. If you require the database files to reproduce the compilation results in the same Quartus II version, you can use the following command-line option to archive a full database:

```
quartus_sh --archive -use_file_set full_db [-revision <revision name>]  
<project name>
```

Formal Verification Support

You cannot use design partitions for incremental compilation if you are creating a netlist for a formal verification tool.

SignalProbe Pins and Engineering Change Orders

ECO and SignalProbe changes are performed only during ECO and SignalProbe compilations. Other compilation flows do not preserve these netlist changes.

When incremental compilation is turned on and your design contains one or more design partitions, partition boundaries are ignored while making ECO changes and SignalProbe signal settings. However, the presence of ECO and/or SignalProbe changes does not affect partition boundaries for incremental compilation. During subsequent compilations, ECO and SignalProbe changes are not preserved regardless of the **Netlist Type** or **Fitter Preservation Level** settings. To recover ECO changes and SignalProbe signals, you must use the Change Manager to re-apply the ECOs after compilation.

For partitions developed independently in separate Quartus II projects, the exported netlist includes all currently saved ECO changes and SignalProbe signals. If you make any ECO or SignalProbe changes that affect the interface to the lower-level partition, the software issues a warning message during the export process that this netlist does not work in the top-level design without modifying the top-level HDL code to reflect the lower-level change. After integrating the **.qxp** partition into the top-level design, the ECO changes will not appear in the Change Manager.



For more information about using the SignalProbe feature to debug your design, refer to the *Quick Design Debugging Using SignalProbe* chapter in volume 3 of the *Quartus II Handbook*. For more information about using the Chip Planner and the Resource Property Editor to make ECOs, refer to the *Engineering Change Management with the Chip Planner* chapter in volume 2 of the *Quartus II Handbook*.

SignalTap II Logic Analyzer in Exported Partitions

You can use the SignalTap II Embedded Logic Analyzer in any project that you can compile and program into an Altera device.

When incremental compilation is turned on, debugging logic is added to your design incrementally and you can tap post-fitting nodes and modify triggers and configuration without recompiling the full design. Use the appropriate filter in the Node Finder to find your node names. Use **SignalTap II: post-fitting** if the netlist type is Post-Fit to incrementally tap node names in the post-fit netlist database. Use **SignalTap II: pre-synthesis** if the netlist type is **Source File** to make connections to the source file (pre-synthesis) node names when you synthesize the partition from the source code.

If incremental compilation is turned off, the debugging logic is added to the design during Analysis and Elaboration, and you cannot tap post-fitting nodes or modify debug settings without fully compiling the design.

For design partitions that are being developed independently in separate Quartus II projects and contain the logic analyzer, when you export the partition, the Quartus II software automatically removes the SignalTap II logic analyzer and related SLD_HUB logic. You can tap any nodes in a Quartus II project, including nodes within **.qxp** partitions. Therefore, you can use the logic analyzer within the full top-level design to tap signals from the **.qxp** partition.

You can also instantiate the SignalTap II megafunction directly in your lower-level design (instead of using an **.stp** file) and export the entire design to the top level to include the logic analyzer in the top-level design.



For details about using the SignalTap II logic analyzer in an incremental design flow, refer to the *Design Debugging Using the SignalTap II Embedded Logic Analyzer* chapter in volume 3 of the *Quartus II Handbook*.

External Logic Analyzer Interface in Exported Partitions

You can use the Logic Analyzer Interface in any project that you can compile and program into an Altera device. You cannot export a partition that uses the Logic Analyzer Interface. You must disable the Logic Analyzer Interface feature and recompile the design before you export the design as a partition.



For more information about the Logic Analyzer Interface, refer to the *In-System Debugging Using External Logic Analyzers* chapter in volume 3 of the *Quartus II Handbook*.

Assignments Made in HDL Source Code in Exported Partitions

Assignments made with I/O primitives or the `altera_attribute` HDL synthesis attribute in lower-level partitions are passed to the top-level design, but do not appear in the top-level `.qsf` file or Assignment Editor. These assignments are considered part of the source netlist files. You can override assignments made in these source files by changing the value with an assignment in the top-level design.

Design Partition Script Limitations

The Quartus II software has some additional limitations related to the design partition scripts described in “[Generating Design Partition Scripts](#)” on page 3–34.

Warnings About Extra Clocks Due to Design Partition Scripts

The generated scripts include applicable clock information for all clock signals in the top-level design. Some of those clocks may not exist in the lower-level projects, so you may see warning messages related to clocks that do not exist in the project. You can ignore these warnings or edit your constraints so the messages are not generated.

Synopsys Design Constraint Files for the TimeQuest Timing Analyzer in Design Partition Scripts

After you have compiled a design using TimeQuest constraints, and the timing assignments option is turned on in the scripts, a separate Tcl script is generated to create an `.sdc` file for each lower-level project. This script includes only clock constraints and minimum and maximum delay settings for the TimeQuest Timing Analyzer.



PLL settings and timing exceptions are not passed to lower-level designs in the scripts. For suggestions on managing SDC constraints between top-level and lower-level projects, refer to the *Best Practices for Incremental Compilation Partitions and Floorplan Assignments* chapter in volume 1 of the *Quartus II Handbook*.

Wildcard Support in Design Partition Scripts

When applying constraints with wildcards, note that wildcards are not analyzed across hierarchical boundaries. For example, an assignment could be made to these nodes: `Top|A:inst|B:inst|*`, where A and B are lower-level partitions, and hierarchy B is a child of A, that is B is instantiated in hierarchy A. This assignment is applied to modules A, B, and all children instances of B. However, the assignment `Top|A:inst|B:inst*` is applied to hierarchy A, but is not applied to the B instances because the single level of hierarchy represented by `B:inst*` is not expanded into multiple levels of hierarchy. To avoid this issue, ensure that you apply the wildcard to the hierarchical boundary if it should represent multiple levels of hierarchy.

When using the wildcard to represent a level of hierarchy, only single wildcards are supported. This means assignments such as `Top|A:inst|*|B:inst|*` are not supported. The Quartus II software issues a warning in these cases.

Derived Clocks and PLLs in Design Partition Scripts

If a clock in the top level is not directly connected to a pin of a lower-level partition, the lower-level partition does not receive assignments and constraints from the top-level pin in the design partition scripts.

This issue is of particular importance for clock pins that require timing constraints and clock group settings. Problems can occur if your design uses logic or inversion to derive a new clock from a clock input pin. Make appropriate timing assignments in your lower-level Quartus II project to ensure that clocks are not unconstrained.

If the lower-level design uses the top-level project framework from the project lead, the design will have all the required information about the clock and PLL settings. Otherwise, if you use a PLL in your top-level design and connect it to lower-level partitions, the lower-level partitions do not have information about the multiplication or phase shift factors in the PLL. Make appropriate timing assignments in your lower-level Quartus II project to ensure that clocks are not unconstrained or constrained with the incorrect frequency. Alternatively, you can manually duplicate the top-level derived clock logic or PLL in the lower-level design file to ensure that you have the correct multiplication or phase-shift factors, compensation delays and other PLL parameters for complete and accurate timing analysis. Create a design partition for the rest of the lower-level design logic for export to the top level. When the lower-level design is complete, export only the partition that contains the relevant logic.

Pin Assignments for GXB and LVDS Blocks in Design Partition Scripts

Pin assignments for high-speed GXB transceivers and hard LVDS blocks are not written in the scripts. You must add the pin assignments for these hard IP blocks in the lower-level projects manually.

Virtual Pin Timing Assignments in Design Partition Scripts

Design partition scripts use `INPUT_MAX_DELAY` and `OUTPUT_MAX_DELAY` assignments to specify inter-partition delays associated with input and output pins, which would not otherwise be visible to the project. These assignments require that the software specify the clock domain for the assignment and set this clock domain to " * ".

This clock domain assignment means that there may be some paths constrained and reported by the timing analysis engine that are not required.

To restrict which clock domains are included in these assignments, edit the generated scripts or change the assignments in your lower-level Quartus II project. In addition, because there is no known clock associated with the delay assignments, the software assumes the worst-case skew, which makes the paths seem more timing critical than they are in the top-level design. To make the paths appear less timing-critical, lower the delay values from the scripts. If required, enter negative numbers for input and output delay values.

Top-Level Ports that Feed Multiple Lower-Level Pins in Design Partition Scripts

When a single top-level I/O port drives multiple pins on a lower-level module, it unnecessarily restricts the quality of the synthesis and placement at the lower-level. This occurs because in the lower-level design, the software must maintain the hierarchical boundary and cannot use any information about pins being logically equivalent at the top level. In addition, because I/O constraints are passed from the top-level pin to each of the children, it is possible to have more pins in the lower level than at the top level. These pins use top-level I/O constraints and placement options that might make them impossible to place at the lower level. The software avoids this situation whenever possible, but it is best to avoid this design practice to avoid these potential problems. Restructure your design so that the single I/O port feeds the design partition boundary and the single connection is split into multiple signals within the lower-level partition.

Restrictions on Megafunction Partitions

The Quartus II software does not support partitions for megafunction instantiations. If you use the MegaWizard™ Plug-In Manager to customize a megafunction variation, the MegaWizard-generated wrapper file instantiates the megafunction. You can create a partition for the MegaWizard-generated megafunction custom variation wrapper file.

The Quartus II software does not support creating a partition for inferred megafunctions (that is, where the software infers a megafunction to implement logic in your design). If you have a module or entity for the logic that is inferred, you can create a partition for that hierarchy level in the design.

The Quartus II software does not support creating a partition for any Quartus II internal hierarchy that is dynamically generated during compilation to implement the contents of a megafunction.

Register Packing and Partition Boundaries

The Quartus II software performs register packing during compilation automatically. However, when incremental compilation is enabled, logic in different partitions cannot be packed together because partition boundaries might prevent cross-boundary optimization. This restriction applies to all types of register packing, including I/O cells, DSP blocks, sequential logic, and unrelated logic. Similarly, logic from two partitions cannot be packed into the same ALM.

I/O Register Packing

Cross-partition register packing of I/O registers is allowed in certain cases where your input and output pins exist in the top-level hierarchy (and the Top partition), but the corresponding I/O registers exist in other partitions.

The following specific circumstances are required for input pin cross-partition register packing:

- The input pin feeds exactly one register.
- The path between the input pin and register includes only input ports of partitions that have one fan-out each.


The following specific circumstances are required for output register cross-partition register packing:

- The register feeds exactly one output pin.
- The output pin is fed by only one signal.
- The path between the register and output pin includes only output ports of partitions that have one fan-out each.

Output pins with an output enable signal cannot be packed into the device I/O cell if the output enable logic is part of a different partition from the output register. To allow register packing for output pins with an output enable signal, structure your HDL code or design partition assignments so that the register and tri-state logic are defined in the same partition.

Bidirectional pins are handled in the same way as output pins with an output enable signal. If the registers that need to be packed are in the same partition as the tri-state logic, you can perform register packing.


The restrictions on tri-state logic exist because the I/O atom (device primitive) is created as part of the partition that contains tri-state logic. If an I/O register and its tri-state logic are contained in the same partition, the register can always be packed with tri-state logic into the I/O atom. The same cross-partition register packing restrictions also apply to I/O atoms for input and output pins. The I/O atom must feed the I/O pin directly with exactly one signal. The path between the I/O atom and the I/O pin must include only ports of partitions that have one fan-out each.


 For more information and examples of cross-partition boundary I/O packing, refer to the *Best Practices for Incremental Compilation Partitions and Floorplan Assignments* 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 or at a command-line prompt. This section provides scripting examples that cover some of the topics discussed in this chapter.

Tcl Scripting and Command-Line Examples

 For information about the `::quartus::incremental_compilation` Tcl package that contains a set of functions for manipulating design partitions and settings related to the incremental compilation feature, refer to `::quartus::incremental_compilation` in Quartus II Help.

 For scripting support information, including design examples and training, refer to the *Quartus II Software Scripting Support* page of the Altera website. For detailed Tcl scripting and command-line information, including design examples, refer to the *Tcl Scripting* and *Command-Line Scripting* chapters in volume 2 of the *Quartus II Handbook*.

Creating Design Partitions

To create a design partition to a specified hierarchy name, use the following command:


```
create_partition [-h | -help] [-long_help] -contents <hierarchy name>
-partition <partition name> ←
```

Table 3-4. Tcl Script Command: create_partition

Argument	Description
-h -help	Short help
-long_help	Long help with examples and possible return values
-contents <hierarchy name>	Partition contents (hierarchy assigned to Partition)
-partition <partition name>	Partition name

Enabling or Disabling Design Partition Assignments During Compilation

To direct the Quartus II Compiler to enable or disable design partition assignments during compilation, use the following command:

```
set_global_assignment -name IGNORE_PARTITIONS <value> ←
```

Table 3-5. Tcl Script Command: set_global_assignment

Value	Description
<i>OFF</i>	The Quartus II software recognizes the design partitions assignments set in the current Quartus II project and recompiles the partition in subsequent compilations depending on their netlist status.
<i>ON</i>	The Quartus II software does not recognize design partitions assignments set in the current Quartus II project and performs a compilation without regard to partition boundaries or netlists.

Setting the Netlist Type

To set the partition netlist type, use the following command:

```
set_global_assignment -name PARTITION_NETLIST_TYPE <value> \
-section_id <partition name> ←
```



The PARTITION_NETLIST_TYPE command accepts the following values: SOURCE, POST_SYNTH, POST_FIT, and EMPTY. For descriptions for these values, refer to “Partition Netlist Type Settings” on page 3-25.

Setting the Fitter Preservation Level for a Post-fit or Imported Netlist

To set the Fitter Preservation Level for a post-fit or imported netlist, use the following command:

```
set_global_assignment -name PARTITION_FITTER_PRESERVATION_LEVEL \
<value> -section_id <partition name> ←
```



The PARTITION_FITTER_PRESERVATION command accepts the following values: NETLIST_ONLY, PLACEMENT, and PLACEMENT_AND_ROUTING. For descriptions for these values, refer to “Fitter Preservation Level Settings” on page 3-27.

Preserving High-Speed Optimization

To preserve high-speed optimization for tiles contained within the selected partition, use the following command:

```
set_global_assignment -name PARTITION_PRESERVE_HIGH_SPEED_TILES_ON
```

Specifying the Software Should Use the Specified Netlist and Ignore Source File Changes

To specify that the software should use the specified netlist and ignore source file changes, even if the source file has changed since the netlist was created, use the following command:

```
set_global_assignment -name PARTITION_IGNORE_SOURCE_FILE_CHANGES ON  
-section_id "<partition name>".
```

Reducing Opening a Project, Creating Design Partitions, and Performing an Initial Compilation

Scenario background: You open a project called AB_project, set up two design partitions, entities A and B, and then perform an initial full compilation.

Example 3-1. AB_project

```
set project AB_project  
  
load_package incremental_compilation  
load_package flow  
project_open $project  
  
# Ensure that design partition assignments are not ignored  
set_global_assignment -name IGNORE_PARTITIONS \ OFF  
  
# Set up the partitions  
create_partition -contents A -name "Partition_A"  
create_partition -contents B -name "Partition_B"  
  
# Set the netlist types to post-fit for subsequent  
# compilations (all partitions are compiled during the  
# initial compilation since there are no post-fit  
# netlists)  
set_partition -partition "Partition_A" -netlist_type POST_FIT  
set_partition -partition "Partition_B" -netlist_type POST_FIT  
  
# Run initial compilation:  
export_assignments  
execute_flow -full_compile  
  
project_close
```

Optimizing the Placement for a Timing-Critical Partition

Scenario background: You have run the initial compilation shown in the example script under [Example 3–1](#). You would like to apply Fitter optimizations, such as physical synthesis, only to partition **A**. No changes have been made to the HDL files. To ensure the previous compilation result for partition **B** is preserved, and to ensure that Fitter optimizations are applied to the post-synthesis netlist of partition **A**, set the netlist type of **B** to **Post-Fit** (which was already done in the initial compilation, but is repeated here for safety), and the netlist type of **A** to **Post-Synthesis**, as shown in the following example:

Example 3–2. AB_project (2)

```
set project AB_project

load_package flow
load_package incremental_compilation
load_package project
project_open $project

# Turn on Physical Synthesis Optimization
set_high_effort_fmax_optimization_assignments

# For A, set the netlist type to post-synthesis
set_partition -partition "Partition_A" -netlist_type POST_SYNTH

# For B, set the netlist type to post-fit
set_partition -partition "Partition_B" -netlist_type POST_FIT

# Also set Top to post-fit
set_partition -partition "Top" -netlist_type POST_FIT

# Run incremental compilation:
export_assignments
execute_flow -full_compile

project_close
```

Generating Design Partition Scripts

To generate design partition scripts, use the following script:

```
# load required package
load_package database_manager

# name and open the project
set project <project_path/project_name>
project_open $project

# generate the design partition scripts
generate_bottom_up_scripts <options>

#close project
project_close
```

- ❓ The options map to the same as those in the Quartus II software in the **Generate Design Partition Scripts** dialog box. For detailed information about each option, refer to *Generate Design Partition Scripts Dialog Box* in Quartus II Help.

Exporting a Partition

To open a project and load the `::quartus::incremental_compilation` package before you use the Tcl commands to export a partition to a **.qxp** that contains both a post-synthesis and post-fit netlist, with routing, use the following script:

```
# load required package
load_package incremental_compilation

# open project
project_open <project name>

# export partition to the .qxp and set preservation level
export_partition -partition <partition name>
-qxp <.qxp file name> -<options>

#close project
project_close
```

Importing a Partition into the Top-Level Design

To import a **.qxp** into a top-level design, use the following script:

```
# load required packages
load_package incremental_compilation
load_package project
load_package flow

# open project
project_open <project name>

#import partition
import_partition -partition <partition name> -qxp <.qxp file> -<options>

#close project
project_close
```

Makefiles

For an example of how to use incremental compilation with a makefile as part of the team-based incremental compilation design flow, refer to the **read_me.txt** file that accompanies the `incr_comp` example located in the `/qdesigns/incr_comp_makefile` subdirectory.

- ② When using a team-based incremental compilation design flow, the **Generate Design Partition Scripts** dialog box can write makefiles that automatically export lower-level design partitions and import them into the top-level design whenever design files change. For more information about the **Generate Design Partition Scripts** dialog box, refer to *Generate Design Partition Scripts Dialog Box* in Quartus II Help.

Conclusion

With the Quartus II incremental compilation feature described in this chapter, you can preserve the results and performance of unchanged logic in your design as you make changes elsewhere. The various applications of incremental compilation enable you to improve your productivity while designing for high-density FPGAs.

Document Revision History

Table 3-6 shows the revision history for this document.

Table 3-6. Document Revision History (Part 1 of 2)

Date	Version	Changes
November 2013	13.1.0	Removed HardCopy device information. Revised information about Rapid Recompile. Added information about functional safety. Added information about flattening sub-partition hierarchies.
November 2012	12.1.0	Added “Turning On Supported Cross-boundary Optimizations” on page 3-21.
June 2012	12.0.0	Removed survey link.
November 2011	11.0.1	Template update.
May 2011	11.0.0	■ Updated “Tcl Scripting and Command-Line Examples”.
December 2010	10.1.0	<ul style="list-style-type: none"> ■ Changed to new document template. ■ Reorganized Tcl scripting section. Added description for new feature: Ignore partitions assignments during compilation option. ■ Reorganized “Incremental Compilation Summary” section.
July 2010	10.0.0	<ul style="list-style-type: none"> ■ Removed the explanation of the “bottom-up design flow” where designers work completely independently, and replaced with Altera’s recommendations for team-based environments where partitions are developed in the same top-level project framework, plus an explanation of the bottom-up process for including independent partitions from third-party IP designers. ■ Expanded the Merge command explanation to explain how it now accommodates cross-partition boundary optimizations. ■ Restructured Altera recommendations for when to use a floorplan. ■ Added “Viewing the Contents of a Quartus II Exported Partition File (.qxp)” section. ■ Reorganized chapter to make design flow scenarios more visible; integrated into various sections rather than at the end of the chapter.
October 2009	9.1.0	<ul style="list-style-type: none"> ■ Redefined the bottom-up design flow as team-based and reorganized previous design flow examples to include steps on how to pass top-level design information to lower-level designers. ■ Moved SDC Constraints from Lower-Level Partitions section to the <i>Best Practices for Incremental Compilation Partitions and Floorplan Assignments</i> chapter in volume 1 of the <i>Quartus II Handbook</i>. ■ Reorganized the “Conclusion” section. ■ Removed HardCopy APEX and HardCopy Stratix Devices section.

Table 3-6. Document Revision History (Part 2 of 2)

Date	Version	Changes
March 2009	9.0.0	<ul style="list-style-type: none"> ■ Split up netlist types table ■ Moved “Team-Based Incremental Compilation Design Flow” into the “Including or Integrating partitions into the Top-Level Design” section. ■ Added new section “Including or Integrating Partitions into the Top-Level Design”. ■ Removed “Exporting a Lower-Level Partition that Uses a JTAG Feature” restriction ■ Other edits throughout chapter
November 2008	8.1.0	<ul style="list-style-type: none"> ■ Added new section “Importing SDC Constraints from Lower-Level Partitions” on page 2-44 ■ Removed the Incremental Synthesis Only option ■ Removed section “OpenCore Plus Feature for MegaCore Functions in Bottom-Up Flows” ■ Removed section “Compilation Time with Physical Synthesis Optimizations” ■ Added information about using a .qxp as a source design file without importing ■ Reorganized several sections ■ Updated Figure 2-10



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

