

# FPGA Incremental Compilation—Divide and Conquer

Jennifer Stephenson  
Sr. Applications Engineer, Altera Corporation  
408-544-6890  
[jstephen@altera.com](mailto:jstephen@altera.com)

## 1 Abstract

For high-density, high-performance FPGA designs, the ability to iterate rapidly during design and debugging stages is critical to meet time-to-market demands. Today's FPGA designers are encountering problems traditionally associated with ASIC designs, especially long place-and-route compilation times and difficulties achieving timing closure. To address these issues, FPGA and EDA vendors are beginning to offer incremental design and compilation capabilities previously available only with ASIC design tools. These capabilities include top-down methodologies that support evolving designs and engineering change orders (ECO), as well as bottom-up design methodologies including team-based design flows. Incremental compilation improves productivity by dramatically reducing design iteration times and preserving results to reach timing closure more easily.

This paper presents incremental compilation methodologies using Mentor Graphics® Precision RTL Synthesis and Altera® Quartus® II software, including user scenarios and design recommendations to maximize the benefits and ensure good quality of results in an incremental compilation flow.

## 2 Incremental Compilation Benefits

Conventionally, a hierarchical design is flattened into a single netlist before logic synthesis and fitting (or placement and routing). The entire design is then recompiled every time there is a change in the design. One reason for this behavior is to obtain optimal quality of results. By processing the entire design, the compiler can perform global optimizations to improve area and performance. However, this leads to long placement and routing times, and typically causes major changes in the placement results and the design's performance even for minor changes in source code or settings. There are many situations in which a more incremental compilation flow is desirable.

Incremental compilation allows a design to be organized into logical partitions for synthesis and fitting.

Design iteration time can be dramatically reduced by focusing new compilations on a particular partition. Optimization techniques, such as physical synthesis, can be targeted to specific design partitions while leaving other blocks untouched. When the partitions in a design are well chosen and placed in the device floorplan, design compilation times can be increased while maintaining or even improving the quality of results.

Preserving performance is another major benefit of incremental compilation technologies. When recompiling the design, the software can use updated code or settings, or reuse and preserve the compilation results for each partition. By compiling only one partition in a design, the results and performance of the remaining partitions remain unchanged. This performance preservation feature allows designers to reach timing closure more efficiently by requiring fewer design iterations.

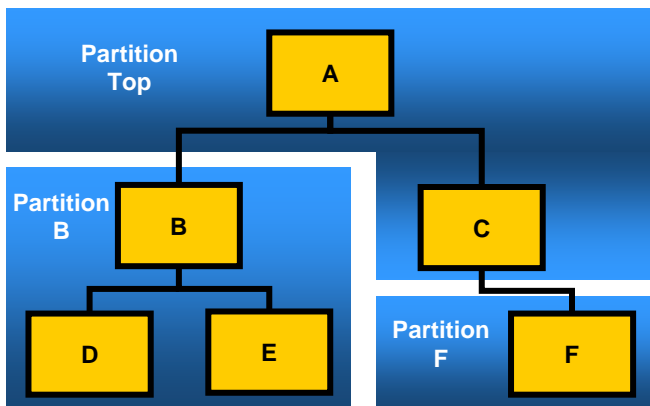
Incremental compilation supports top-down design methodologies, where one designer manages the project for the entire design, as well as bottom-up design methodologies in which each design block can be developed independently. Bottom-up methodologies include team-based design flows where design partitions are created by team members in another location or by third-party intellectual property (IP) providers. Separate partitions are integrated after placement optimization without the partitions affecting each other in the final design. Incremental flows can also be used to compile and optimize some design partitions when other partitions are missing or incomplete.

Altera's Quartus II software delivers incremental compilation technology that provides the productivity improvements demanded by today's FPGA designers. Using the Quartus II incremental compilation feature along with the Precision Synthesis software can reduce placement and routing time, preserve design performance to achieve timing closure more easily, and facilitate hierarchical and team-based design methodologies.

### 3 Design Partitions

It is common design practice to create modular or hierarchical designs in which entities are designed separately and then instantiated in a higher-level project to form a complete design. When using incremental compilation technology, the design hierarchy is mapped to Design Partitions that are treated separately during compilation to allow incremental compilation. Each entity in a design is not automatically considered a design partition for incremental compilation; one or more design hierarchies below the top level must be designated as design partitions.

When a partition is declared, every hierarchy within that partition becomes part of the same partition. When new partitions are created for hierarchies within an existing partition, the logic within the new lower-level partition is no longer part of the higher-level partition. Figure 1 shows an example design hierarchy where module F has been designated as Partition F. Partition B is defined to contain module B and its submodules D and E. Partition Top contains the top-level module A as well as module C because it has not been assigned to any other partition.



**Figure 1. Partitions in a Design Hierarchy**

Creating partitions prevents the synthesis tool and placement and routing tool from performing optimizations across partition boundaries, but it also allows incremental compilation by enabling each partition to separately be synthesized and placed.

Design partitions for incremental compilation are logical partitions, different from physical regions in the device floorplan, which specify a size and location. A logical design partition does not refer to a physical section of the device and is not used to directly control logic placement. A logical design partition sets up a virtual boundary between design hierarchies so that each partition is compiled separately and no logical optimizations can occur between them. It is

recommended to also assign each design partition to a physical region in the device floorplan to improve quality of results with incremental compilation.

### 4 Recommendations for Creating Design Partitions

When planning a design, keep in mind the size and scope of each partition, and how likely it is that different parts of the design change as the design develops. Since cross-boundary optimizations cannot occur when using partitions, the quality of results or performance of the design may decrease as the number of partitions increases. So, while having more partitions allows for greater reduction in compilation time, limit the number of partitions to prevent degradation of the quality of results.

Register the input and output ports of each partition, whenever possible, to avoid any delay on signals that cross partition boundaries. At the very least, either the inputs or the outputs should be registered.

Try to minimize the number of paths that cross partition boundaries to ease the timing closure process.

Do not use tri-state signals or bidirectional ports on partition boundaries unless the port is directly connected to an I/O pin and the tri-state logic is all contained in the same partition. Boundary tri-states must be pushed through the design hierarchy to take advantage of tri-state drivers on the output pins, and this optimization cannot happen across partition boundaries.

### 5 Timing Budgeting

Any paths that cross between partitions are not optimized as an entire path during synthesis. In addition, if lower-level partitions are optimized and imported to the top level, Quartus II software cannot optimize the placement of the entire path. When the synthesis and placement software optimizes the input and output logic in each partition, there is no information about the logic connected to that port in the next partition. During placement, if the logic in one partition is placed far away from logic in another partition, the routing delay between the logic could lead to problems meeting the timing requirements.

One way to reduce these effects is to ensure input and output ports of the partitions are registered whenever possible, as discussed above.

When ports are not registered, some manual timing budgeting may be required to ensure that Quartus II software correctly optimizes the input and output logic

in each partition. For each unregistered timing path that crosses between partitions, make timing assignments on the corresponding I/O path in each partition to constrain both ends of the path to the budgeted timing delay.

Virtual Pin assignments can be used in Quartus II software to indicate pins that do not feed external I/O ports in the Altera device. This is required if a lower-level design has more I/O ports than the number of external I/O pins on the device. Virtual pins can be used to make assignments during manual timing budgeting. For example, if an I/O port feeds a register in another partition, assign the Virtual Pin to the LogicLock™ region that represents the placement of that second partition. Doing so will provide the software more information about the placement of the logic fed by the partition I/O port. By assigning location and timing constraints to Virtual Pins, the quality of the timing budget can be improved.

## 6 Resource Balancing

When using design partitions for incremental compilation, each partition is synthesized separately, with no data about the resources used in other partitions. This means that device resources could be overused in the individual partitions during synthesis, and thus the design may not fit in the target device when the partitions are merged. In a bottom-up design flow where designers optimize their lower-level designs and export them to a top-level design, Quartus II software also places and routes each partition separately. In some cases, partitions can use conflicting resources when combined at the top level.

To avoid these effects, manual resource balancing across partitions may be required.

### 6.1 RAM and DSP Blocks

In the regular synthesis flow, when DSP blocks or RAM blocks are overused, the Precision Synthesis software performs resource balancing and converts some of the logic into regular logic cells — logic elements (LEs) or adaptive logic modules (ALMs). Without data about resources used in other partitions, it is possible for the logic in each separate partition to maximize the use of a particular device resource, such that the design does not fit once all the partitions are merged. In this case, perform manual resource balancing using synthesis options to control inference of megafunctions that use the DSP or RAM blocks. The Quartus II MegaWizard® Plug-In Manager can also be used to customize RAM or DSP megafunctions to use regular logic instead of dedicated hardware blocks.

Altera recommends using a LogicLock region for each partition to minimize the chance that logic in more than one partition uses the same logic resource. However, there are situations in which partition placement may still cause conflicts at the top level. This can occur in cases where a partition is designed one way in a lower-level design (for example, using an M-RAM memory block) and then instantiated in two different ways in the top level (for example, using an M-RAM block for one instance and an M4K block for the other). In this case, using a post-fit netlist only with no placement information allows the software to refit the logic.

### 6.2 Global Routing Signals

Global routing signals can cause conflicts in placement and routing when multiple projects are imported into a top-level design.

Quartus II software automatically promotes high fan-out signals to use global routing resources available in the device. It is possible for lower-level partitions to use the same global routing resources, causing conflicts at the top level. In addition, logic array block (LAB) placement depends on whether the inputs to the LCELLs within the LAB are using a global clock signal. Therefore, problems can occur if a design does not use a global signal in the lower-level design, but does use a global signal in the top-level design.

To avoid these problems, the project leader should determine which partitions will use particular global routing signals. Then each designer of a lower-level partition can assign the appropriate global signals manually, and prevent other signals from using global routing resources.

Use the Global Signal assignment set to a value of either On or Off to place a signal on a global routing line or to prevent the signal from using a global routing line. If the automatic global promotion performed in the Fitter is disabled, turn off the Auto Global Clock and Auto Global Register Control Signals. Alternately, to avoid problems when importing, direct the Fitter to discard the placement and routing of the imported netlist, by setting the Fitter preservation level property of the partition to Netlist Only. With this option, the Fitter re-assigns all the global signals for this particular partition when compiling the top-level design.

## 7 Creating a Design Floorplan

Once the design is partitioned, designers should assign each partition to a physical location on the device. The simplest way to create a floorplan for a partitioned design is to create a LogicLock location constraint for each partition (including the top-level partition).

Floorplan location planning is very important for a design that uses incremental compilation because it helps avoid the situation that arises when the fitter is directed to place or replace a portion of the design in an area of the device where most resources have already been claimed. In this case, the placement of the post-fit netlists of other partitions forces the fitter to place the new or modified partition in the empty parts of the device. There are two immediate disadvantages to this situation. First, the fitter must work harder due to the higher number of physical constraints, and therefore compilation time typically increases. Second, the quality of results often decreases because the placement of the target partition is now scattered throughout the device.

## 8 Using Precision Synthesis with Quartus II Incremental Compilation

In an incremental design flow, ensure that different parts of the design do not affect each other. When using Precision Synthesis, create separate netlists for each partition in the design so that the advantages of Quartus II incremental compilation flows are realized. If the entire design is in one EDIF netlist file, changes in one partition affect other partitions because of optimizations across design hierarchies and possible node name changes when the design is resynthesized.

To maintain the incremental flow, create separate partitions only for modules, entities, or existing netlist files. In addition, each module or entity should have a separate design file. If two different modules are in the same design file but are defined as being part of different partitions, incremental synthesis cannot be maintained because both partitions must be recompiled when one of the modules in the file is changed.

To create a separate EDIF netlist file for each partition, set up a separate Precision RTL implementation or project for each lower-level partition and for the top-level design. Creating different implementations for each partition allows switching between partitions without leaving the current project file. Alternately, a separate project can be created for each partition in a bottom-up or team-based design flow.

Create black-box instantiations of lower-level modules in higher-level implementations or projects. The top-level design implements the top-level entity and instantiates wrapper files representing each subdesign. The wrapper files define the port interfaces, but not the implementation of each lower-level module.

The Precision RTL Synthesis software creates a Tcl file for each EDIF file that provides Quartus II software with

the appropriate constraints and information to set up a project. Depending on the design methodology, one Quartus II project can be created for all EDIF netlists (in a top-down flow), or a separate Quartus II project for each EDIF netlist (in a bottom-up flow).

In a top-down compilation design flow, create design partition assignments and floorplan location assignments for each design partition within a single Quartus II project. This methodology is easy to manage and can provide the best quality of results. Use a bottom-up design flow when each partition must be optimized separately, such as in certain team-based design flows. To perform a bottom-up compilation in Quartus II software, create separate Quartus II projects and integrate each design partition into the top-level design using the incremental compilation export and import features to maintain placement and routing results.

The following steps provide a general compilation flow for using Precision Synthesis with incremental compilation in Quartus II software:

1. Determine which hierarchical blocks are to be treated as separate partitions in the design.
2. Create a project with multiple implementations (or create multiple projects) in Precision Synthesis, one for each partition in the design.
3. Create black box instantiations for lower-level partitions in the top-level design, and in any design module that includes an instantiation of a lower-level partition.
4. Disable I/O pad insertion (Add IO Pads) in the implementations for lower-level partitions.
5. Compile and synthesize each implementation or each project in Precision Synthesis, and make constraints as in the regular design flow.
6. Import the EDIF netlist and the Tcl file for each partition into the Quartus II software.
  - Note that by combining all EDIF netlists in one top-down Quartus II project, constraints made only for lower-level implementations are not passed to Quartus II software. Ensure that appropriate constraints are made in the top-level Precision Synthesis project, or in the Quartus II project.
7. Set up the Quartus II project(s) to use the incremental compilation feature.
  - a. Turn on Full Incremental Compilation.
  - b. Perform Analysis and Elaboration to identify the hierarchy of EDIF netlists in Quartus II software.

- c. Create Design Partition assignments for each EDIF netlist.
  - d. Create a design floorplan using LogicLock location assignments to specify a placement region for each Design Partition.
8. Compile the design in Quartus II software and preserve the compilation results using the appropriate Netlist Type setting for incremental compilation.
  9. When design or synthesis optimization changes are made to part of the design, resynthesize only the changed partition to generate the new EDIF netlist and Tcl file. Do not resynthesize the implementations or projects for the unchanged partitions.
  10. Import the new EDIF netlist and Tcl file into Quartus II software and recompile the design using incremental compilation.
3. In the Quartus II Design Partitions Window, change the Netlist Type to Post-Fit for the remaining partitions that should be preserved. Set the Fitter Preservation Level to either Placement or Placement and Routing. For the partition that contains the fix, change the Netlist Type to Source File to indicate that the partition should be compiled from the source file. When using Precision Synthesis, the source file for Quartus II software is the EDIF netlist. The Source File setting is optional because the Quartus II software automatically recompiles partitions if changes are detected in the source file/netlist.
  4. Click Start Compilation to incrementally compile the fixed HDL code. During this compilation, the Partition Merge stage merges the modified partition with the post-fit netlists of the remaining partitions. This compilation should take much less time than the initial full compilation.
  5. Run the simulation again to ensure that the bug is fixed, and use the Timing Analyzer report to ensure that timing results have not degraded.

## 9 User Scenarios

This section describes user scenarios where incremental compilation can be beneficial, and how to use the software in each scenario. All scenarios assume the design is set up in multiple Precision Synthesis implementations, and that the project is set up to use the full incremental compilation flow in Quartus II software. For details about using the incremental compilation feature in Quartus II software, refer to reference [1]. For details about creating projects in Precision Synthesis, refer to reference [2].

### 9.1 *Changing a Source File for One of Multiple Partitions in a Top-Down Compilation Flow*

In this scenario, a lengthy complete compilation of a design consisting of multiple partitions has just been performed. An error is found in the HDL source file for one partition and is being fixed. Because the design currently meets its timing requirements and the fix is not expected to affect timing performance, only the affected partition should be compiled and the remainder of the design can be preserved.

Perform the following steps to update the single source file:

1. Apply and save the fix to the HDL source file.
2. Resynthesize the one affected Precision Synthesis implementation and generate one new EDIF netlist file.

### 9.2 *Optimizing the Placement for One of Multiple Partitions in a Top-Down Compilation Flow*

In this scenario, a lengthy full compilation of a design consisting of multiple partitions has just been performed. The Timing Analyzer report shows that the clock timing requirement has not been met. After some analysis, it is determined that timing closure can be achieved if placement can be improved for one particular partition. Various optimization settings will be tried to improve the performance. Since these techniques all involve significant compilation time, they will be applied to only the partition in question.

Perform the following steps to enable an advanced optimization setting for just one partition:

1. In the Quartus II Design Partitions Window, set the Netlist Type to Post-Synthesis for the partition in question. This causes the partition to be compiled with the new fitter optimization settings during the next compilation, but does not require Quartus II software to reread and elaborate the EDIF netlist.
2. For the remaining partitions (including the top-level entity), set the Netlist Type to Post-Fit. Set the Fitter Preservation Level to Placement to allow for the most flexibility during routing. To reduce compilation time further, use the Placement and Routing setting. These partitions are preserved during the next compilation.
3. Apply the desired optimization settings.

4. Click Start Compilation to incrementally compile the design with the new settings. During this compilation, the Partition Merge stage merges the post-synthesis netlist of the critical partition with the post-fit netlists of the remaining partitions. The Fitter then refits only the one partition using the new optimization settings. Since the effort is reduced as compared to the initial full compilation, the compilation time is also reduced.

### 9.3 Team-Based Bottom-Up Design Flow

In this scenario, a project consisting of several lower-level subdesigns is split into projects that are implemented separately by different designers. The top-level project instantiates each of these subdesigns. Subdesign designers want to optimize their designs independently and pass on the results to the project leader.

As the project leader in this scenario, perform the following steps to prepare the design for a successful bottom-up design methodology and import the files sent in by the designs of each subdesign:

1. Create a Quartus II project that will ultimately contain the full implementation of the entire design and turn on Full incremental compilation.
  2. Make design partition assignments for each subdesign EDIF netlist, and set the Netlist Type for each design partition that will be imported to Empty in the Design Partitions window.
  3. Create LogicLock regions for the part of the design to be implemented in the top-level project.
  4. When the project is set up correctly, pass design constraints to the designers working on the subdesigns (such as the device and speed grade, global signal constraints, any resource allocation, and timing requirements). Communicate the proposed overall floorplan scheme to the designers to ensure that no two regions occupy the same location in the device. Tcl scripts can be used to pass these assignments.
  5. After the Quartus II Exported Partition file for each subdesign is obtained from the other designers in the team, on the Project menu, click Import Design Partition and specify the partition in the top-level project that is represented by the subdesign Quartus II Exported Partition file.
2. Make LogicLock region assignments and global assignments (including clock settings) as specified by the project leader.
  3. Make Virtual Pin assignments for ports representing connections to core logic instead of external device pins in the top-level module.
  4. Make floorplan location assignments to the Virtual Pins assuring placement in the corresponding regions as determined by the top-level module. This provides the Fitter with more information about the timing constraints between modules. Alternately, apply timing I/O constraints to the paths that connect to virtual pins.
  5. When the required compilation results are achieved, on the Project menu, click Export Project as Design Partition.
  6. Provide the Quartus II Exported Partition file to the project lead.

As a designer of a lower-level subdesign in this scenario, perform the following steps to successfully export the design:

1. Create a new Quartus II project for the subdesign and turn on Full incremental compilation.

## 10 Conclusion

Altera's Quartus II incremental compilation technology delivers the productivity improvement designers need by offering shorter design iteration times, providing unprecedented features to preserve design performance, and supporting team-based design flows. Use this technology to shorten design iteration time and dramatically reduce overall development time. The performance preservation and team-based flows supported through incremental compilation save even more design time by allowing you to reach timing closure more efficiently with fewer design iterations.

Using incremental compilation with Mentor Graphics' Precision Synthesis software provides a complete incremental design flow providing excellent quality of results.

## 11 References

- [1] [Quartus II Incremental Compilation](#) in Volume 1 of the [Quartus II Handbook](#).
- [2] [Mentor Graphics Precision RTL Synthesis Support](#) in Volume 1 of the [Quartus II Handbook](#).



101 Innovation Drive  
San Jose, CA 95134  
(408) 544-7000  
<http://www.altera.com>

Copyright © 2006 Altera Corporation. All rights reserved. Altera, The Programmable Solutions Company, the stylized Altera logo, specific device designations, and all other words and logos that are identified as trademarks and/or service marks are, unless noted otherwise, the trademarks and service marks of Altera Corporation in the U.S. and other countries. All other product or service names are the property of their respective holders. Altera products are protected under numerous U.S. and foreign patents and pending applications, maskwork rights, and copyrights. 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 Corporation. 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.