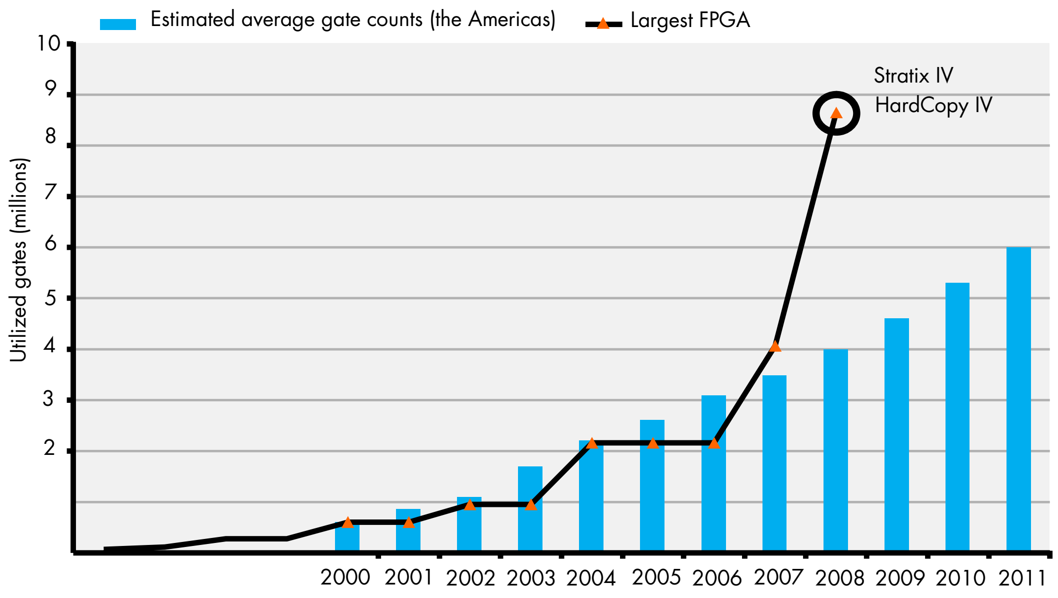
Increasing Productivity With Quartus II Incremental Compilation

Introduction

Designers are creating FPGAs that continue to increase in logic density and performance, yet their time-to-market pressures are becoming even more demanding. Computing power is not increasing as rapidly to maintain compilation times for synthesis, placement, and routing. For example, the number of logic cells in Altera® devices has grown by 35 times and the number of memory bits by over 100 times in the last ten years, yet computing power has grown only 10 times. Larger and higher performance designs in larger FPGAs are leading to longer compilation times.

Today's FPGA designers are expressing concerns about compilation times and achieving timing closure that previously were associated only with ASIC designs. In 2008, Altera's largest Stratix® IV and HardCopy® IV designs are expected to reach more than twice the average gate count for ASIC designs, as shown in Figure 1. Without Altera's efforts to reduce compilation time, these large designs would take more than a full day to compile with every design change, leading to inefficiency and wasted design time.

Figure 1. FPGAs Exceed Size of Average ASIC Design (1)



Note:
(1) Data based on Gartner Dataquest report, 11/21/07.

The ability to iterate rapidly during the FPGA design and debugging stages is critical for success. Designers want to reduce the compilation time for each design iteration, and feel confident that they can preserve their timing closure results to reduce the number of design iterations required to complete the design. To address these concerns, a proven approach in ASIC design is now taking hold for FPGAs. FPGA and EDA vendors are offering incremental design and compilation capabilities previously available only with ASIC design tools. These capabilities include top-down methodologies that support design reuse, evolving designs, and engineering change orders, as well as bottom-up design methodologies that include team-based design flows.

Altera's Quartus® II design software delivers incremental compilation to provide the productivity improvements demanded by today's FPGA designers. This white paper describes how an incremental compilation flow improves design productivity for high-density, high-performance FPGAs.

WP-01062-1.0

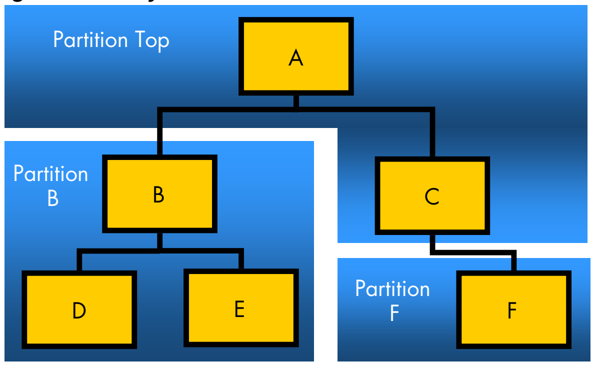
Conventional "Flat" Compilation vs. Incremental Compilation

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 that by processing the entire design, the compiler can perform global optimizations to improve resource utilization and timing performance. However, compiling the entire design can lead to longer-than-expected compilation times for small design changes. In addition, designers can be frustrated to find that a change in one part of the design affects timing closure for a separate part of the design. Designers can use a flat compilation flow successfully for small designs, such as designs in CPLDs or low-density FPGAs, when the timing requirements are met easily with a single push-button compilation. A flat design is satisfactory when compilation time and preserving results for timing closure are not concerns.

Using an incremental flow preserves the results and performance for unchanged logic in a design as changes are made elsewhere. It can reduce design compilation time dramatically, allowing more design iterations per day and achieving timing closure more efficiently. Incremental design flows also support easier design reuse, and facilitate team-based design methodologies that allow designers to create and optimize design blocks independently in today's multiple-geography design environments. Incremental compilation is recommended for large designs and high device densities, as well as for designs that require high performance relative to the speed of the device architecture.

When using incremental compilation, the design hierarchy is mapped to design partitions that are treated separately during compilation to allow incremental functionality. Each entity or instance in a design is not automatically considered a design partition; the designer must designate one or more design hierarchies below the top level as design partitions for incremental compilation. 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 2 shows an example design hierarchy in which instances B and F have been designated as design partitions. Partition B includes its sub-instances D and E. The default partition, "Top," contains the top-level block A as well as instance C because it has not been assigned to any other partition.

Figure 2. Partitions in a Design Hierarchy



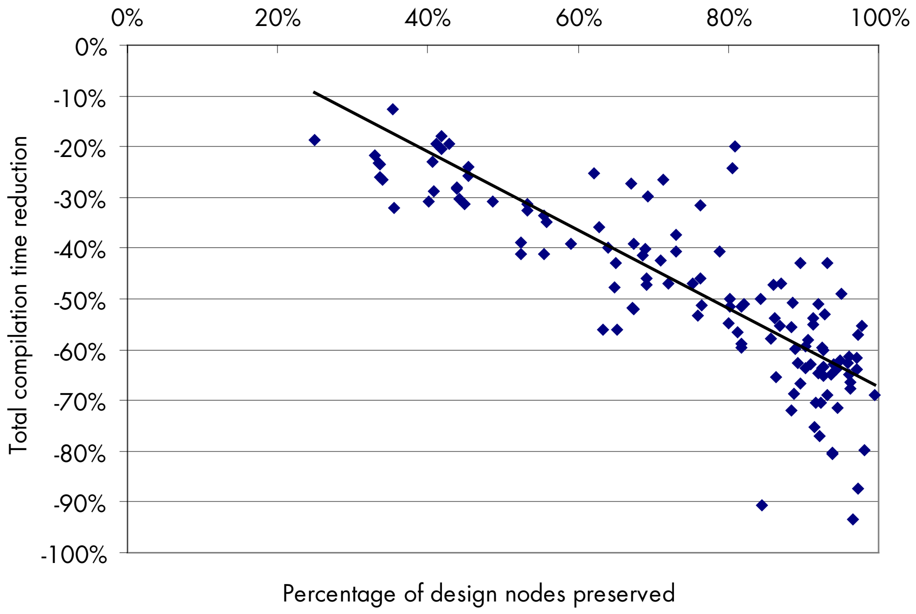
Creating partitions prevents the software from performing logic optimizations across partition boundaries during synthesis or placement, which also allows incremental behavior by enabling each partition to be synthesized or placed and routed independently at any time.

How Incremental Compilation Benefits Productivity

Incremental compilation allows a design hierarchy to be separated into logical partitions for synthesis and fitting, as described in the previous section. When recompiling the design, the software can use updated HDL code and settings, or reuse and preserve the compilation results for each partition. Design iteration time is reduced by focusing the compilation effort on a particular partition, while partitions that do not change are not recompiled. Optimization techniques, such as physical synthesis, can also be targeted to specific design partitions while leaving other logic blocks untouched.

The benchmark tests in Figure 3 show that when recompiling after making a design change with 80 percent or more of the design nodes preserved, the average compilation time reduction ranges between roughly 50 and 70 percent. This reduction means that two or three design iterations can be performed in the same amount of time as one flat compilation.

Figure 3. Fitter Time Reduction When Preserving Unchanged Parts of the Design



Preserving performance is another major benefit of incremental compilation technologies. By compiling only specific partitions in a design, the timing performance of the remaining partitions remains unchanged. Performance preservation allows designers to reach timing closure more efficiently by requiring fewer design iterations.

Incremental compilation supports both top-down and bottom-up design methodologies. In top-down flows, different designers or intellectual property (IP) providers can create and verify HDL code separately, and then one designer manages the final Quartus II project for the entire design. With bottom-up design flows, individual designers or IP providers can complete the placement and routing optimization of their design in separate projects and then export each lower-level project into one top-level project. These methodologies support team-based design flows in which design partitions are created by team members in another location or by third-party companies. Separate partitions can be integrated without the partitions affecting each other. Incremental flows can also be used to compile and optimize some design partitions when other partitions are missing or incomplete.

Design Scenarios Where Incremental Flows Can Help

Incremental compilation can be beneficial in different stages of the design flow. These scenarios explain how incremental compilation helps improve design productivity in various ways.

Reducing Compilation Time When Changing a Source File

Incremental compilation allows the source code for one partition to be updated without having to recompile other parts of the design. For example, this capability can help when an error is found in an HDL source file after compilation. When the fix is isolated to one part of the design and is not expected to affect timing performance for other design blocks, it is desirable to compile only the affected code and preserve the rest of the design.

If the design is split into partitions for incremental compilation, the designer can update the single source file in the Quartus II software as follows. First, apply and save the fix to the HDL source file. Set the software to reuse the post-fit netlists for partitions that did not change, and allow the software to automatically recompile partitions with changes detected in the source files. Doing so results in a much shorter compilation time than reprocessing the entire design, and preserves the performance for the unchanged blocks, which reduces any need for additional timing closure effort.

Optimizing Results for Part of the Design Before Adding Other Logic

Incremental compilation also can be used to optimize one set of partitions in isolation and then lock the placement to preserve the results while the rest of the design is completed. For example, a partition can be created for some IP that comes with instructions to perform optimization before incorporating the rest of the custom logic. This flow works best if each critical path is contained within a single partition so that there are no dependencies between partitions.

For this scenario, only the partition that represents the timing-critical IP block is compiled with any required extra optimizations turned on. Assign other design blocks as “empty” partitions so their logic is not compiled. After timing closure is achieved for the critical partition, preserve its contents and placement with a post-fit netlist. Now timing closure for this logic block does not have to be revisited, and the logic can be compiled for the remaining partitions.

Debugging Incrementally With an In-System Logic Analyzer

Debugging incrementally can be a powerful feature of an incremental design flow. This flow can reduce compilation times when adding an internal logic analyzer to debug a design, or when modifying the configuration of the logic analyzer without modifying the logic design or its placement. Quartus II incremental compilation preserves the results of the original design, and adds the Altera SignalTap® II Logic Analyzer without recompiling the original source code.

It is not necessary to create any design partitions to debug incrementally with the SignalTap II logic analyzer; the analyzer automatically acts as its own separate design partition for incremental compilation. Set the software to preserve the post-fit results for all design partitions, including the default Top partition. Set up the analyzer to probe post-fitting node names in the design netlist, and the Quartus II Fitter can add the SignalTap II logic to the post-fit netlist during the next compilation without modifying the existing design results.

Implementing a Team-Based Design Environment and Bottom-Up Design Flow

A project consisting of several low-level designs can be split into multiple projects that are implemented separately by different designers. In this scenario, the top-level project instantiates each of the low-level designs and designers of each block can optimize their designs independently to ensure timing closure before passing on the compilation results to the project leader.

The project leader in this scenario first creates a project that will ultimately contain the entire design, creates empty design partitions for each subdesign, and passes design constraints to the designers working on the low-level designs (typically including floorplan constraints to avoid resource location conflicts). Each team member optimizes their design independently and verifies the timing requirements, then exports their partition along with its placement information. The project lead imports the information for each design block into the top-level project and compiles the entire design.

If all the timing-critical paths are isolated within a single partition, the top-level design will meet its timing requirements with no further effort. If cross-partition paths do not meet the timing requirements, the project lead can provide additional information to the low-level designers to help them optimize each design. The Quartus II software includes a feature that automatically generates scripts to pass information about the overall design to the separate designers of low-level blocks, such as locations for ports that connect partitions.

Follow Guidelines for Setting Up an Incremental Design

Incremental compilation flows may require more up-front planning than full flat compilations. For example, the source code or design hierarchy may have to be structured to ensure that logic is grouped correctly for optimization. It is easier to implement the correct logic grouping early in the design cycle than to restructure the code later. Once the design is partitioned, designers may want to assign each partition to a physical location on the device to create a design floorplan. Poor partition or floorplan assignments can hurt design area utilization and performance, making timing closure more difficult. Incremental compilation generally requires that designers be more rigorous about following good design practices than flat compilations.

When planning a design, the designer should 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. Often-changing logic should be kept separate from fixed parts of the design. The design hierarchy should isolate timing-critical logic inside a single partition, ideally with registered port boundaries, so the software can effectively optimize each partition independently.

Using the guidelines provided in Altera’s documentation when planning a design helps ensure good results. As FPGAs get larger and more complex, following good design practices becomes more important for all design flows. Adhering to the recommended hierarchical synchronous design practices makes designs more robust and easier to debug. Using an incremental compilation flow may add additional steps and requirements to a project, but can provide significant benefits in design productivity by preserving the performance of critical blocks and reducing compilation times.

Conclusion

Incremental compilation methodologies continue to take hold in the FPGA design community as FPGA designs increase in size and performance. Altera’s Quartus II incremental compilation technology delivers the productivity improvement designers need by reducing design compilation times by an average of 60 percent, preserving design performance, and supporting team-based design flows. With some design planning and partitioning, this technology shortens design iteration time and considerably reduces development time. The performance preservation and team-based flows supported through incremental compilation allow designers to reach timing closure more efficiently with fewer design iterations, and achieve faster time to market.

Further Information

- *Quartus II Incremental Compilation for Hierarchical and Team-Based Design* in Volume 1 of the *Quartus II Handbook*:
  www.altera.com/literature/hb/qts/qts_qii51015.pdf
- *Best Practices for Incremental Compilation Partitions and Floorplan Assignments* in Volume 1 of the *Quartus II Handbook*:
  www.altera.com/literature/hb/qts/qts_qii51017.pdf
- *Leveraging the 40-nm Process Node to Deliver the World’s Most Advanced Custom Logic Devices*:
  www.altera.com/literature/wp/wp-01058-stratix-iv-40nm-process-node-custom-logic-devices.pdf

Acknowledgements

- Jennifer Stephenson, MTS Applications Engineer, Software Applications Engineering, Altera Corporation
- Scott Brissenden, Advanced Software Engineer, Performance Analysis Group, Software and IP Engineering, Altera Corporation
- Albert Chang, Senior Product Marketing Engineer, Software Marketing, Altera Corporation

101 Innovation Drive
San Jose, CA 95134
www.altera.com