Increasing Design Functionality with Partial and Dynamic Reconfiguration in 28-nm FPGAs

WP-01137-1.0 White Paper

The density of FPGAs has grown with each process node shrink. Compared to previous generations of FPGAs, the extra density, coupled with features such as reconfiguration, enables designers to add to or change the functionality of these devices. FPGAs provide the perfect platform for accommodating improvements and design changes in functionality because of the reconfiguration feature. However, for some applications such as the 100G-Optical Transport Network (OTN) multiplexing transponders (muxponders), this full reconfiguration feature may not be good enough. Because the muxponder application requires changes in logic without disrupting the entire system and stopping the flow of data, a need for a partial reconfiguration feature comes into play. Altera® Stratix® V FPGAs support partial reconfiguration along with many other features to address the 100G-OTN muxponder application and other applications.

Introduction

Partial reconfiguration is the ability to reconfigure part of the FPGA while the rest of the device continues to work. The biggest benefit users can derive from this feature is reduced device count. Partial reconfiguration improves logic density by removing the need to implement functions that do not operate simultaneously in the FPGA. Using smaller devices or a reduced number of devices improves system cost and lowers power consumption. Important applications for this technology include reconfigurable communication systems and high-performance computing platforms.

In static random access memory (SRAM)-based FPGA architectures, all user-programmable features are controlled by memory cells that are volatile and must be configured on power-up. These memory cells, also called configuration random access memory (CRAM), contain the functions of the logic cells, routing, power-up conditions of registers, I/O voltage standards, and various other aspects of the FPGA.

The configuration memory is programmed via a bitstream, which contains the instructions for the control block and all the data for the configuration memory. This bitstream is programmed into the FPGA through a host using one of various configuration schemes, such as the fast passive parallel (FPP) configuration FPP x16, or through any available communication protocol such as the PCI Express® (PCIe®), Serial RapidIO® and Gigabit Ethernet (GbE) standards.

An SRAM-based FPGA can be partially programmed with a partially-generated bitstream that contains the instructions for the control block and the data for the configuration memory. Supporting such functionality requires innovation in the silicon and intelligence built into the control block to handle the partially generated bitstream file. The FPGA fabric needs the ability to clear the existing functionality in the logic cells and program new functionality, while the rest of the fabric remains functional. Stratix V FPGAs offer this partial reconfiguration capability through innovation and intelligence built into the silicon fabric and Quartus® II software.

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

Applications

Partial reconfiguration is suitable for designs with many permutations that do not operate simultaneously and hence, can share the same resources on the FPGA. In such systems, one section of the FPGA continues to operate, while the other section is reconfigured for new functionality. Partial reconfiguration is suited for applications that require continuous operations because it provides an advantage over full reconfiguration with the ability to reconfigure a portion of the device. One example is the 100G-OTN system, where the user can reconfigure each channel on the fly with a partially-generated bitstream and change the functionality of the chosen port. Another example of partial reconfiguration is in high-performance computing systems where frequent algorithm updates are required for different platforms being serviced. Partial reconfiguration supports compression and encryption of the partial bitstream for military applications such as creating enhanced design security applications, where the partial region is used to decrypt the incoming bitstream, encrypt a new bitstream, and configure the rest of the device. Partial reconfiguration allows the system to maintain real-time links while other modules within the FPGA are being configured.

Partial Reconfiguration Methodology

To implement a partial reconfiguration design successfully, up-front planning is required. The designer needs to identify the blocks needed to reconfigure in real time (partial blocks) and blocks that remain operational (static blocks). To be successful in implementing a design using partial reconfiguration, strictly follow these guidelines:

- Follow the synthesis guideline for designing the modules.
- Identify the partial blocks and floorplan them using the incremental compilation methodology and the LogicLock™ flow.
- Module boundaries of all the partial blocks and their variants should not change, otherwise the entire FPGA may need to be reconfigured.
- Resets between partial blocks and static blocks should not be shared.
- Logic within the static blocks that depend on the state of the partial block should have proper handshaking logic to deal with the availability and non-availability of the partial blocks.

Altera offers two flows for creating partial reconfiguration designs:

- Design partition-based flow—In this flow, the user reserves a space on the device and the region or space can be configured while the rest of the design is running.
- Engineering change order (ECO)-based flow—This is a flow for users to make minimal changes to design logic while the rest of the design is running.

These two flows leverage incremental compilation and Altera's state-of-the art Chip Planner tool in the Quartus II software. These features and tools help plan the design for partial reconfiguration.

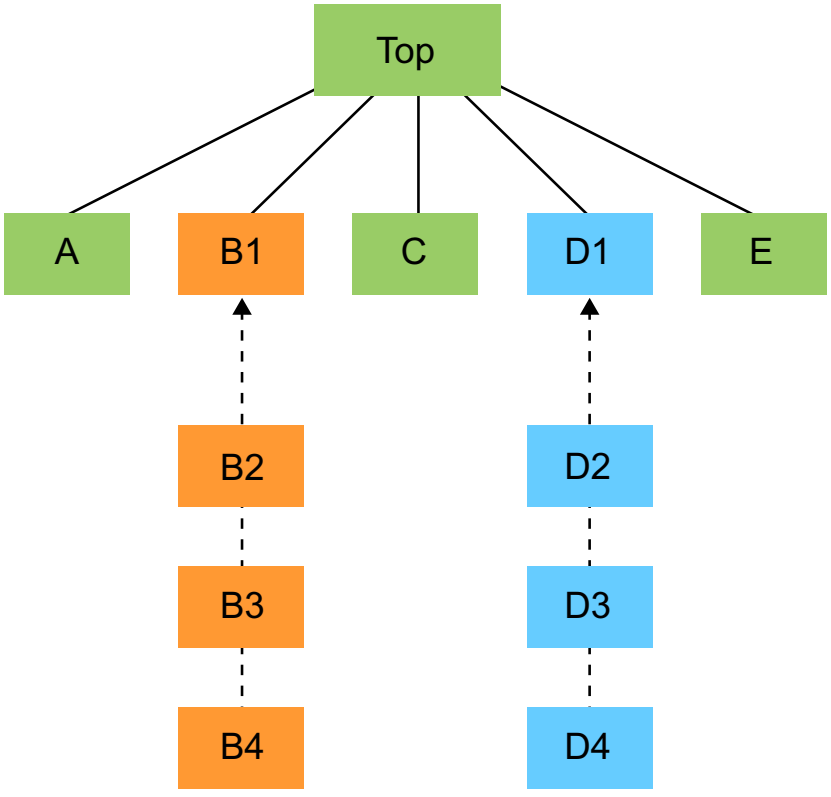
Implementation in the Quartus II Software

Partial reconfiguration improves effective logic density by removing the need to place functions that do not operate simultaneously in the FPGA. Altera has simplified the partial reconfiguration process with a design flow built on top of the proven incremental compilation design and LogicLock flows in the Quartus II design software. To incorporate partition reconfiguration into the Stratix V design flow, plan the design hierarchy by following these steps:

1. Identify the design blocks to replace on subsequent device configurations, and plan to implement each of those design blocks in a separate source file.
2. Create a top-level Quartus II project with top-level hierarchy that instantiates each design block.
3. Create different source design files that describe the different logic functions that can be implemented for each reconfigurable design block when reconfiguring the device. The port definitions or hierarchical boundaries for any reconfigurable design block must be the same for each of its functions, so that the block can be reconfigured without requiring changes to any other logic in the top-level design.

   For example, the top-level design in Figure 1 instantiates several design blocks, including two design blocks, B and D, that will be changed using partial reconfiguration. Blocks B and D each have four different logic functions that may be required in the FPGA, so there are 16 combinations of design logic that can be implemented in the FPGA. If the initial compilation uses instances B1 and D1 and produces a full-chip programming file, then eight partial programming files must also be generated so that B and D can be reconfigured again at any time with a different logic function.

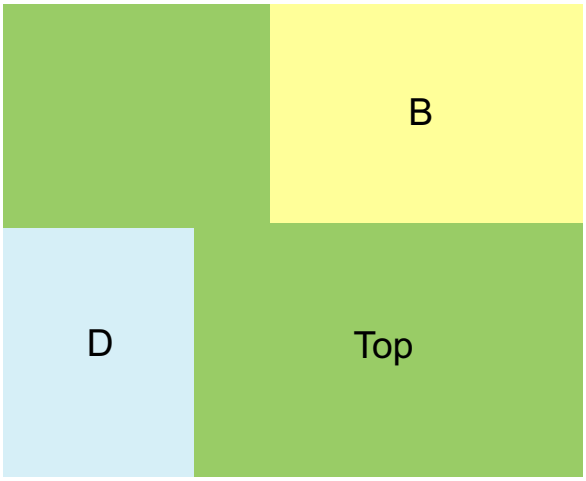
Figure 1. Top-Level Design with Two Reconfigurable Design Partitions



4. In the Quartus II software, identify each of the reconfigurable design blocks as a design partition. This process ensures that each design block can be compiled and configured incrementally, by preserving the hierarchical boundaries and using a separate Quartus II compilation netlist for each partition. There is no need to partition the entire design; just the design blocks that will be compiled and reconfigured incrementally. For the example in Figure 1, create partitions for blocks B and D. A top-level partition called "Top" automatically includes the rest of the design that is not defined as separate partitions.
5. Constrain each reconfigurable partition to a physical region in the device floorplan by assigning each partition to a reserved LogicLock region with a fixed size and location. The LogicLock region defines the physical boundaries of the FPGA that can be reconfigured.

   Figure 2 shows a schematic representation of the FPGA floorplan, where partition B has been constrained to the top-right corner of the device while partition D has been constrained to the lower-left corner. The logic in the top-level design and non-partitioned logic blocks A, C, and E are part of partition Top that can be placed anywhere else in the device.

Figure 2. FPGA Floorplan with Reconfigurable Design Partitions Assigned to LogicLock Regions



When working with the Quartus II software, designers can create configuration files for each of the source files that represent different functions in each reconfigurable design partition. The device can be reconfigured with any combination of these files.

While developing the design, each design file for each design partition must meet its timing requirements when the complete design is compiled. To confirm that each version of each design partition meets timing requirements with the same implementation for the rest of the design, set **Netlist Type** for the Top partition to **Post-Fit** to preserve the placement and routing results. Then the top-level design can be compiled with each source file for each design partition and the timing results confirmed. In this example, there are 16 possible combinations of design blocks that must be verified.

Design Recommendations

The boundaries of the design partitions can impact the design's quality of results, because creating partitions prevents the compiler from performing logic optimizations across partition boundaries. 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 timing-critical connections as possible inside a partition allows the designer to effectively apply optimizations to that partition, while leaving the rest of the design unchanged. In addition, avoid constant partition inputs and outputs because, to maintain incremental behavior, the software cannot use the constants to optimize logic on either side of the partition boundary.

For more information on making design partitions and using an incremental design flow, refer to the *Quartus II Incremental Compilation for Hierarchical and Team-Based Design* chapter in the *Quartus II Handbook*.

For more design guidelines to ensure good quality of results, and suggestions on making design floorplan assignments with LogicLock regions, refer to the *Best Practices for Incremental Compilation Partitions and Floorplan Assignments* chapter in the *Quartus II Handbook*.

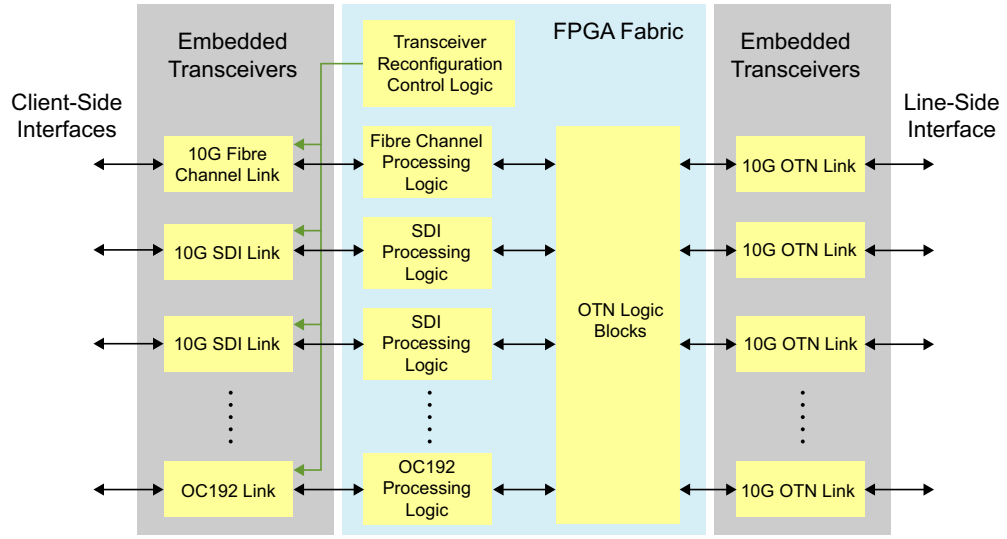
Partial Reconfiguration Feature Application

An example application that significantly benefits from the Stratix V partial reconfiguration feature is the 100G-OTN muxponder application. The OTN muxponder aggregates different protocol traffic such as Fibre Channel, SDI, Ethernet, and SONET from multiple client-side interface ports into a common OTN transport infrastructure. Importantly, the system should have the ability to dynamically receive multiple protocol traffic on each of these client-side interfaces. Stratix V FPGAs offer a flexible one-chip solution satisfying these requirements.

Stratix V FPGAs have multiple built-in embedded transceiver blocks that provide an aggregate bandwidth of over 500 Gbps and perform the physical coding sublayer (PCS) and physical medium attachment (PMA) functionality of multiple high-speed serial protocols. These dynamically-reconfigurable hard macros save valuable FPGA fabric resources, eliminate the verification effort, and enable the designer to dynamically switch the transceivers to support multiple protocols and data rates.

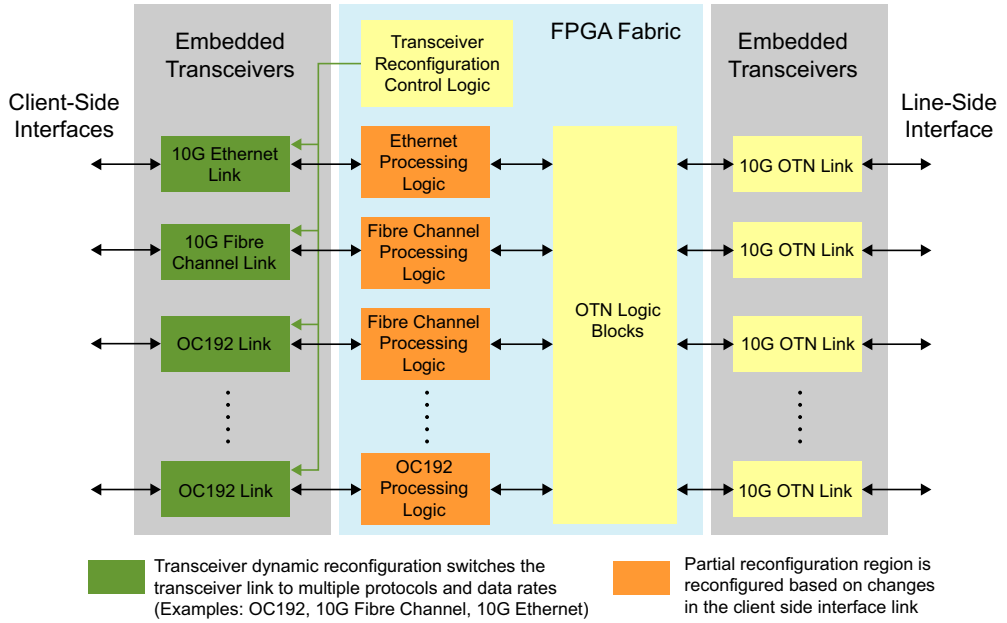
Figure 3 shows the typical OTN muxponder system with multiple client side protocol interfaces and OTN line side interface. Each client side interface link requires a protocol processing logic block in the FPGA fabric. The partial reconfiguration feature offers the flexibility to reconfigure the protocol processing logic block based on the changes in the client side interface link.

Figure 3. Block Diagram of a Typical OTN Muxponder System



For example, Figure 4 shows the protocol processing logic blocks that are partially reconfigured (shown in orange) due to new protocol traffic received on the client's interface. With this feature, a designer can implement an OTN muxponder application within a single Stratix V FPGA, significantly reducing system cost that would otherwise be incurred in a multi-chip solution.

Figure 4. Dynamically Reconfigured Blocks in the OTN Muxponder System



Quartus II software provides a simplified design flow and user interface to implement transceiver reconfiguration. Using this flow, the designer can generate multiple unique transceiver configurations that can be stored in on-chip or off-chip memory. The Quartus II software provides reconfiguration control logic that reads the transceiver settings from the memory and handles the sequences to complete the transceiver dynamic reconfiguration.

When designing a OTN muxponder application, follow these steps:

1. Keep the same transceiver-FPGA fabric interface for all the transceiver configurations so that the input signals to the partial reconfiguration block from the transceiver interface remain the same. The megafunction that generates transceiver instances provides the options to select all the interface signals needed for different protocol configurations.
2. Create a separate partition for the static block, and create a LockLock region outside the partial reconfiguration partition. Place the reconfiguration control logic generated by the Quartus II software and the memory files that contain the different transceiver settings into that static block partition.
3. For the protocol-specific processing logic associated with each transceiver channel (in orange in Figure 4), create different partitions for the protocol-processing logic and a LogicLock region to reserve space in the device floorplan.
4. Create a design file for each protocol-processing logic and specify the different protocol-specific logic files for the channel during compilation. For example, as shown in Figure 4, each channel can be reconfigured to the Fibre Channel, Ethernet, or SDI protocol. For each channel, compile three different design files that implement the corresponding protocol-processing logic function.
5. After reconfiguring the transceivers to a new data rate, download the bitstream that contains the protocol-processing logic corresponding to this new protocol traffic.

Implementation of Configuration Schemes

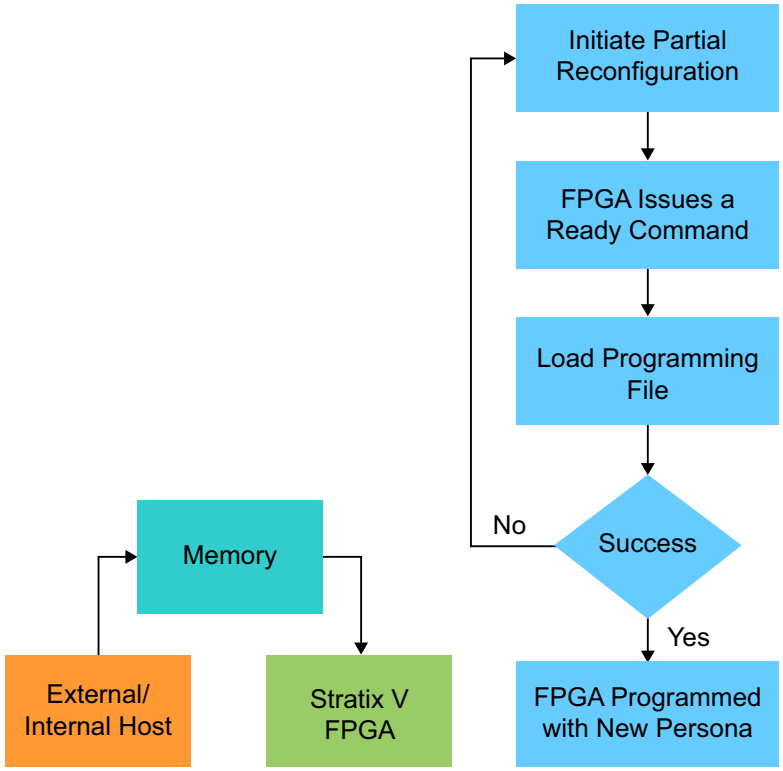
Partial reconfiguration supports two modes of operation: an external host and an internal host. Designers can use any other configuration scheme to perform partial reconfiguration using an internal host. The internal host can be user logic or a Nios® II processor. For external host operation, only the FPP x16 mode is supported.

FPP x16 Configuration with an External Host

For previous generations of FPGAs, FPP is supported in 8-bit data width. For Stratix V devices, FPP is supported in 8-, 16-, and 32-bits data width. FPP with 16-bits data width (FPP x16) supports partial reconfiguration. Using the FPP configuration scheme, the host can be either a MAX II CPLD or a CPU that controls the configuration process. The configuration data is stored in a configuration memory such as CFI parallel flash. During configuration, the host fetches the data from the memory and configures the FPGA.

When FPP x16 is used with partial reconfiguration, the same configuration host (either the MAX II CPLD or the CPU) can act as the partial reconfiguration host. As illustrated in Figure 5, the partial reconfiguration data can be stored in the configuration memory just like in a regular FPP configuration. During partial reconfiguration, the host initiates the partial reconfiguration cycle, reads the data from the configuration memory, and transmits the data to the FPGA.

Figure 5. FPP x16 Sequence



Configuration with an Internal Host

Partial reconfiguration using an internal host is similar to partial reconfiguration in the FPP x16 configuration scheme, except partial reconfiguration is initiated internally from the FPGA fabric. The configuration data can be stored in an embedded memory block or an external memory. Similar to FPP x16, the internal host (either user logic or a Nios II processor) initiates the partial reconfiguration cycle, fetches the data, and sends the data directly to the FPGA core through the FPGA fabric.

Conclusion

Partial reconfiguration provides designers with the benefits of reduced device count, reduced power, and overall reduced cost. When used in conjunction with dynamic reconfiguration of transceivers, it provides a flexible solution to implement high-bandwidth applications such as 100G-OTN muxponders and many other applications. Stratix V FPGAs support partial reconfiguration with their state-of-the-art reconfigurable fabric and the proven technology of incremental compile and LogicLock in Quartus II software. With this solution, Altera has implemented an easy-to-use flow that does not require the designer to have intricate knowledge of the FPGA architecture.

Further Information

- Stratix V FPGAs: Built for Bandwidth: www.altera.com/products/devices/stratix-fpgas/stratix-v/stxv-index.jsp
- Literature: Stratix V Devices: www.altera.com/products/devices/stratix-fpgas/stratix-v/literature/stv-literature.jsp
- Stratix V FPGAs: Ultimate Flexibility Through Partial and Dynamic Reconfiguration www.altera.com/products/devices/stratix-fpgas/stratix-v/overview/partial-reconfiguration/stxv-part-reconfig.html
- Webcast: "Learn About Easier-to-Use Partial Reconfiguration Flow": www.altera.com/products/devices/stratix-fpgas/stratix-v/overview/partial-reconfiguration/stxv-part-reconfig.html
- White paper: *Enabling 100-Gbit OTN Muxponder Solutions on 28-nm FPGAs*: www.altera.com/literature/wp/wp-01126-stxv-100g-otn-muxponder.pdf
- *Quartus II Handbook: Quartus II Incremental Compilation for Hierarchical and Team-Based Design*: www.altera.com/literature/hb/qts/qts_qii51015.pdf
- *Quartus II Handbook: Best Practices for Incremental Compilation Partitions and Floorplan Assignments*: www.altera.com/literature/hb/qts/qts_qii51017.pdf

Acknowledgements

- Ajay Jagtiani, Software Technical Marketing Manager, Altera Corporation
- Jennifer Stephenson, Member of Technical Staff, Software Applications Engineering, Altera Corporation
- Sridhar Krishnamurthy, Member of Technical Staff, High-Speed Applications Engineering, Altera Corporation
- Noor Hazlina Ramly, Senior Component Applications Engineer, Components and Software Applications Engineering, Altera Corporation

Document Revision History

Table 1 shows the revision history for this document.

Table 1. Document Revision History

Date	Version	Changes
July 2010	1.0	Initial release.