Vivado Design Suite User Guide

Partial Reconfiguration

UG909 (v2015.4) November 18, 2015

XILINX
ALL PROGRAMMABLE™

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
11/18/2015	2015.4	Updated device support in Overview and Design Considerations. Updated PR license checking process in Partial Reconfiguration Licensing. Added information to Avoiding Deadlock about handshaking across the RM. Updated how I/O is treated during reconfiguration in I/O Rules. Updated Clearing Bitstreams to clarify that the clearing bitstream is required to load a new RM. Added *Partial Reconfiguration of a Hardware Accelerator with Vivado Design Suite for Zynq-7000 AP SoC Processor* (XAPP1231) to Appendix A, Additional Resources and Legal Notices.
09/30/2015	2015.3	Updated device compatibility for UltraScale devices in Design Considerations. Updated Design Criteria to include the new PR Decoupler IP and guidelines for connecting an RM flop to an I/O buffer. Added Reading Design Constraints section. Clarified `lock_design` command in Preserving Implementation Data. Updated Create a Floorplan for the Reconfigurable Region. Revisions to Table 3-4, Pblock Commands and Properties. Added "Important" note about `HD.PARTPIN_RANGE` to Context Property Examples:. Added `report_clock_utilization` to Reporting. Updated Bitstream Generation. Updated Packing Logic and Design Instance Hierarchy. Added "Recommended" to Reconfigurable Partition Interfaces. Added section on Avoiding Deadlock. Updated `ROUTING` entry in Table 5-1, SNAPPING_MODE Property Values for 7 Series Devices. Updated Global Clocking Rules. Added figures and a table of DRCs to Partial Reconfiguration Checklist for UltraScale Device Designs. Removed Known Issues on running `phys_opt_design` in UltraScale. Updated SEM IP support information and added UltraScale encryption use cases to Known Limitations.

Date	Version	Revision
06/24/2015	2015.2	Updated device compatibility for 2015.2 release. Added additional consideration to Design Requirements and Guidelines. Added IOB minimum PU to Creating Pblocks for UltraScale Devices. Corrected available values for `BITSTREAM.CONFIG.PERSIST` in Overview. Updated ChipScope references to Vivado Logic Analyzer. Added section Blanking Bitstreams Recommended for 7 series and Zynq-7000 Family Partial Reconfiguration Designs.
04/01/2015	2015.1	Added to the Overview section updated device support information and a link to a QuickTake Video on using Partial Reconfiguration with UltraScale™ devices. Updated device support information in the Design Requirements and Guidelines section. Updated section on Timing Constraints to note new Vivado Design Suite capability to run cell level timing reports. Updated section Apply Reset After Reconfiguration to add information regarding designs that use the DRP interface of the 7 series XADC component. Added a table showing SNAPPING_MODE property values for 7 series devices. Also enhanced description of the SNAPPING_MODE property. See the section Automatic Adjustments for Reconfigurable Partition Pblocks. Added "Important" note at the bottom of section Automatic Adjustments for PU on PBlocks. Added a note to Global Clocking Rules section. To the Partial Reconfiguration Checklist for UltraScale Device Designs: Added information on DRC rule restrictions to the "Recommended Clocking Networks" item. Added an item for SSI technology. Added section on Bitstream Type Definitions. Added section Partial Reconfiguration through ICAP for Zynq Devices. Revisions to Table 7-1, PCIe® block and Reset Locations Supporting PR, by Device. Added section Formatting BIN Files for Delivery to Internal Configuration Ports. Added item regarding Engineering Silicon (ES) for UltraScale devices page 106.

Table of Contents

Revision History 2

Chapter 1: Introduction

Overview 7
Introduction to Partial Reconfiguration 8
Terminology 9
Design Considerations 12
Partial Reconfiguration Licensing 16

Chapter 2: Common Applications

Overview 17
Networked Multiport Interface 17
Configuration by Means of Standard Bus Interface 19
Dynamically Reconfigurable Packet Processor 21
Asymmetric Key Encryption 22
Summary 23

Chapter 3: Vivado Software Flow

Overview 24
Partial Reconfiguration Commands 25
Partial Reconfiguration Constraints and Properties 31
Apply Reset After Reconfiguration 38
Software Flow 41
Tcl Scripts 46

Chapter 4: Design Considerations and Guidelines for All Xilinx Devices

Overview 47
Design Hierarchy 47
Partition Pin Placement 51
Active-Low Resets and Clock Enables 51
Decoupling Functionality 52
Black Boxes 53
Effective Approaches for Implementation 54

Defining Reconfigurable Partition Boundaries 55
Avoiding Deadlock 56
Design Revision Checks 57
Simulation and Verification 57

Chapter 5: Design Considerations and Guidelines for 7 Series and Zynq Devices

Overview 58
Design Elements Inside Reconfigurable Modules 58
Global Clocking Rules 59
Creating Pblocks for 7 Series Devices 61
Using High Speed Transceivers 70
Partial Reconfiguration Design Checklist (7 Series) 70

Chapter 6: Design Considerations and Guidelines for UltraScale Devices

Overview 74
Design Elements Inside Reconfigurable Modules 74
Creating Pblocks for UltraScale Devices 75
Global Clocking Rules 78
I/O Rules 79
Using High Speed Transceivers 80
Partial Reconfiguration Checklist for UltraScale Device Designs 80

Chapter 7: Configuring the Device

Overview 87
Configuration Modes 88
Bitstream Type Definitions 89
Partial Reconfiguration through ICAP for Zynq Devices 93
Accessing the Configuration Engine through the MCAP 94
Formatting BIN Files for Delivery to Internal Configuration Ports 96
Summary of BIT Files for UltraScale Devices 97
System Design for Configuring an FPGA 98
Partial BIT File Integrity 99
Configuration Frames 101
Configuration Time 102
Configuration Debugging 102

Chapter 8: Known Issues and Limitations

Known Issues 106
Known Limitations 108

Appendix A: Additional Resources and Legal Notices

Xilinx Resources 110
Solution Centers 110
References 110
Training Resources 112
Please Read: Important Legal Notices 112

Introduction

Overview

Partial Reconfiguration allows for the dynamic change of modules within an active design. This flow requires the implementation of multiple configurations which ultimately results in full bitstreams for each configuration, and partial bitstreams for each Reconfigurable Module.

The number of configurations required varies by the number of modules that need to be implemented. However, all configurations use the same top-level, or static, placement and routing results. These static results are exported from the initial configuration, and imported by all subsequent configurations using checkpoints.

This guide:

- Is intended for designers who want to create a partially reconfigurable FPGA design.
- Assumes familiarity with FPGA design software, particularly Xilinx® Vivado® Design Suite.
- Has been written specifically for Vivado Design Suite Release 2015.4. This release supports the following products:
  - 7 Series devices: This release supports Partial Reconfiguration for all Virtex®-7, Kintex®-7, Artix®-7, and Zynq®-7000 All Programmable SoC devices.
  - UltraScale™ devices: This release includes device support for the following:
    - Place and route and bitstream generation is enabled for all production devices except the KU025 and VU440, which will be supported in a future version of Vivado design suite.
    - Bitstream generation is disabled by default for ES2 devices, but place and route can still be performed.
- Describes Partial Reconfiguration as implemented in the Vivado toolset.

VIDEO: *For an overview of the Vivado Partial Reconfiguration solution in 7 series devices, see the Vivado Design Suite QuickTake Video: Partial Reconfiguration in Vivado.*

VIDEO: *For an overview of the Vivado Partial Reconfiguration solution in UltraScale devices, see the Vivado Design Suite QuickTake Video: Partial Reconfiguration for UltraScale.*

Introduction to Partial Reconfiguration

FPGA technology provides the flexibility of on-site programming and re-programming without going through re-fabrication with a modified design. Partial Reconfiguration (PR) takes this flexibility one step further, allowing the modification of an operating FPGA design by loading a partial configuration file, usually a partial BIT file. After a full BIT file configures the FPGA, partial BIT files can be downloaded to modify reconfigurable regions in the FPGA without compromising the integrity of the applications running on those parts of the device that are not being reconfigured.

Figure 1-1 illustrates the premise behind Partial Reconfiguration.

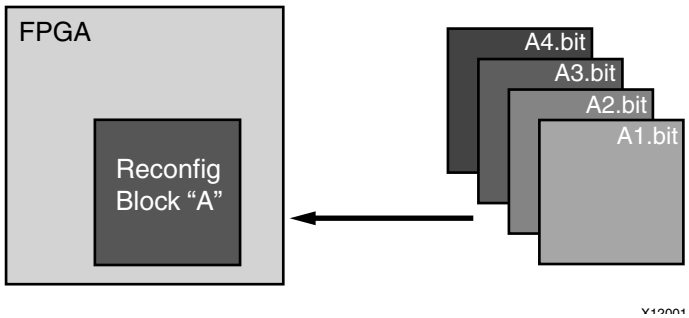

Figure 1-1: **Basic Premise of Partial Reconfiguration**

As shown, the function implemented in Reconfig Block A is modified by downloading one of several partial BIT files, `A1.bit`, `A2.bit`, `A3.bit`, or `A4.bit`. The logic in the FPGA design is divided into two different types, reconfigurable logic and static logic. The gray area of the FPGA block represents static logic and the block portion labeled Reconfig Block "A" represents reconfigurable logic. The static logic remains functioning and is unaffected by the loading of a partial BIT file. The reconfigurable logic is replaced by the contents of the partial BIT file.

There are many reasons why the ability to time multiplex hardware dynamically on a single FPGA is advantageous. These include:

- Reducing the size of the FPGA required to implement a given function, with consequent reductions in cost and power consumption
- Providing flexibility in the choices of algorithms or protocols available to an application
- Enabling new techniques in design security
- Improving FPGA fault tolerance

- Accelerating configurable computing

In addition to reducing size, weight, power and cost, Partial Reconfiguration enables new types of FPGA designs that would be otherwise impossible to implement.

Terminology

The following terminology is specific to the Partial Reconfiguration feature and is used throughout this document.

Bottom-Up Synthesis

Bottom-Up Synthesis is synthesis of the design by modules, whether in one project or multiple projects. Bottom-Up Synthesis requires that a separate netlist is written for each Partition, and no optimizations are done across these boundaries, ensuring that each portion of the design is synthesized independently. Top-level logic must be synthesized with black boxes for Partitions.

Configuration

A Configuration is a complete design that has one Reconfigurable Module for each Reconfigurable Partition. There might be many Configurations in a Partial Reconfiguration FPGA project. Each Configuration generates one full BIT file as well as one partial BIT file for each Reconfigurable Module.

Configuration Frame

Configuration frames are the smallest addressable segments of the FPGA configuration memory space. Reconfigurable frames are built from discrete numbers of these lowest-level elements. In Xilinx devices, the base reconfigurable frames are one element (CLB, BRAM, DSP) wide by one clock region high. The number of resources in these frames vary by device family.

Internal Configuration Access Port (ICAP)

The Internal Configuration Access Port (ICAP) is essentially an internal version of the SelectMAP interface. For more information, see the *7 Series FPGAs Configuration User Guide* (UG470) [Ref 7] or the *UltraScale Architecture Configuration User Guide* (UG570) [Ref 8].

Media Configuration Access Port (MCAP)

The MCAP is dedicated link to the ICAP from one specific PCIe® block per UltraScale device. This entry point can be enabled when configuring the Xilinx PCIe IP.

Partial Reconfiguration (PR)

Partial Reconfiguration is modifying a subset of logic in an operating FPGA design by downloading a partial bitstream.

Partition

A Partition is a logical section of the design, user-defined at a hierarchical boundary, to be considered for design reuse. A Partition is either implemented as new or preserved from a previous implementation. A Partition that is preserved maintains not only identical functionality but also identical implementation.

Partition Pin

Partition pins are the logical and physical connection between static logic and reconfigurable logic. Partition pins are automatically created for all Reconfigurable Partition ports.

Processor Configuration Access Port (PCAP)

The Processor Configuration Access Port (PCAP) is similar to the Internal Configuration Access Port (ICAP) and is the primary port used for configuring a Zynq-7000 AP SoC device. For more information, see the *Zynq-7000 All Programmable Technical Reference Manual* (UG585) [Ref 9].

Programmable Unit (PU)

In the UltraScale architecture, the minimum required resources for reconfiguration. The size of a PU varies by resource type. Because adjacent sites share a routing resource (or Interconnect tile) in the UltraScale architecture, a PU is defined in terms of pairs.

Reconfigurable Frame

Reconfigurable frames (in all references other than "configuration frames" in this guide) represent the smallest reconfigurable region within an FPGA. Bitstream sizes of reconfigurable frames vary depending on the types of logic contained within the frame.

Reconfigurable Logic

Reconfigurable Logic is any logical element that is part of a Reconfigurable Module. These logical elements are modified when a partial BIT file is loaded. Many types of logical components can be reconfigured such as LUTs, flip-flops, block RAM, and DSP blocks.

Reconfigurable Module (RM)

A Reconfigurable Module (RM) is the netlist or HDL description that is implemented within a Reconfigurable Partition. Multiple Reconfigurable Modules will exist for a Reconfigurable Partition.

Reconfigurable Partition (RP)

Reconfigurable Partition (RP) is an attribute set on an instantiation that defines the instance as reconfigurable. The Reconfigurable Partition is the level of hierarchy within which different Reconfigurable Modules are implemented. Tcl commands such as `opt_design`, `place_design` and `route_design` detect the `HD.RECONFIGURABLE` property on the instance and process it correctly.

Static Logic

Static logic is any logical element that is not part of a Reconfigurable Partition. The logical element is never partially reconfigured and is always active when Reconfigurable Partitions are being reconfigured. Static logic is also known as top-level logic.

Static Design

The Static design is the part of the design that does not change during partial reconfiguration. The static design includes the top level and all modules not defined as reconfigurable. The static design is built with static logic and static routing.

Design Considerations

Partial Reconfiguration (PR) is an expert flow within the Vivado Design Suite. Prospective customers must understand the following requirements and expectations before embarking on a PR project.

Design Requirements and Guidelines

- Partial Reconfiguration requires the use of Vivado 2013.3 or newer.
  - Partial Reconfiguration is supported in the ISE Design Suite as well. Use the ISE Design Suite for Partial Reconfiguration only with Virtex-6, Virtex-5 and Virtex-4 devices. See the ISE *Partial Reconfiguration User Guide* (UG702) [Ref 10] for more information.
- Device support in Vivado Design Suite 2015.4:
  - 7 Series: All Artix-7, Kintex-7, Virtex-7, and Zynq-7000 SoC devices.
  - UltraScale:
    - Implementation support and bitstream generation for all production silicon UltraScale devices except for the KU025 and VU440.
    - Implementation support only (no bitstream generation) for ES2 versions of the above devices. ES2 support is unofficial and should be used for development purposes only.
- PR is supported through Tcl or by command line only; there is no project support at this time.
- Floorplanning is required to define reconfigurable regions, per element type.
  - For greatest efficiency, and to use the `RESET_AFTER_RECONFIG` feature with 7 series devices, vertically align to frame/clock region boundaries.
  - Horizontal alignment rules also apply. See Create a Floorplan for the Reconfigurable Region in Chapter 3 for more information.
- Bottom-up synthesis (to create multiple netlist files) and management of Reconfigurable Module netlist files is the responsibility of the user.
  - Any synthesis tool can be used. Disable I/O insertion to create Reconfigurable Module netlists.
  - Vivado Synthesis uses the out-of-context Module Analysis flow for Reconfigurable Module synthesis.
- Standard timing constraints are supported, and additional timing budgeting capabilities are available if needed.

- A unique set of design rule checks (DRCs) has been established to help ensure successful design completion.
- A PR design must consider the initiation of Partial Reconfiguration as well as the delivery of partial BIT files, either within the FPGA or as part of the system design.
- The 2015.1 release of the Vivado Design Suite introduced support for a new Partial Reconfiguration Controller IP. This customizable IP manages the core tasks for partial reconfiguration in a 7 series, Zynq-7000, or UltraScale device. The core receives triggers from hardware or software, manages ing and decoupling tasks, fetches partial bitstreams from memory locations, and delivers them to the ICAP.
  More information on the PR Controller IP is available on the Xilinx website.
- A Reconfigurable Partition must contain a super set of all pins to be used by the varying Reconfigurable Modules implemented for the partition. It is expected that this results in unused inputs or outputs for some modules and is designed into the flexibility of the PR solution. The unused inputs are left inside the module. Drive outputs to a constant if this is an issue for your design.
- Black boxes are supported for bitstream generation. See Black Boxes in Chapter 4 for details about how to tie off ports with constant values.
- For user reset signals, determine if the logic inside the RM is level or edge sensitive. If the reset circuit is edge sensitive (as it may be in some IP such as FIFOs), then the RM reset should not applied until after configuration is complete.

Design Performance

Performance metrics vary from design to design, and the best results are achieved if you follow the Hierarchical Design techniques documented in the *Hierarchical Design Methodology Guide* (UG748) [Ref 11]. This documents was created for the ISE Design Suite, but the methodologies contained therein apply for the Vivado Design Suite. You can find additional design recommendations in the *UltraFast Design Methodology Guide for the Vivado Design Suite* (UG949) [Ref 12].

However, the additional restrictions that are required for silicon isolation are expected to have an impact on most designs. The application of Partial Reconfiguration rules, such as routing containment, exclusive placement, and no optimization across reconfigurable module boundaries, means that the overall density and performance is lower for a PR design than for the equivalent flat design. The overall design performance for PR designs varies from design to design, based on factors such as the number of reconfigurable partitions, the number of interface pins to these partitions, and the size and shape of Pblocks.

Any potential Partial Reconfiguration design must have extra timing slack and resource overhead before considering this solution. See the Building Up Implementation Requirements, page 54 section for more information on evaluating a design for PR.

Design Criteria

- Some component types can be reconfigured and some cannot.

  For 7 series devices, the component rules are as follows:

  - Reconfigurable resources include CLB, BRAM, and DSP component types as well as routing resources.
  - Clocks and clock modifying logic cannot be reconfigured, and therefore must reside in the static region.
    - Includes BUFG, BUFR, MMCM, PLL, and similar components
  - The following components cannot be reconfigured, and therefore must reside in the static region:
    - I/O and I/O related components (ISERDES, OSERDES, IDELAYCTRL)
    - Serial transceivers (MGTs) and related components
    - Individual architecture feature components (such as BSCAN, STARTUP, ICAP, XADC.)

  For UltraScale devices, the list of reconfigurable component types is more extensive:

  - CLB, BRAM, and DSP component types as well as routing resources
  - Clocks and clock modifying logic, including BUFG, MMCM, PLL, and similar components
  - I/O and I/O related components (ISERDES, OSERDES, IDELAYCTRL)

    Note: The types of changes for I/O components is limited. See I/O Rules in Chapter 6 for more information.
  - Serial transceivers (MGTs) and related components
  - PCIe, CMAC, Interlaken, and SYSMON blocks
  - Bitstream granularity of these new components require that certain rules are followed. For example, partial reconfiguration of I/O require that the entire bank, plus all clocking resources in that frame are reconfigured together.
  - Only the configuration components (such as BSCAN, STARTUP, ICAP, and FRAME_ECC) must remain in the static portion of the design.

- Global clocking resources to Reconfigurable Partitions are limited, depending on the device and on the clock regions occupied by these Reconfigurable Partitions.

- IP restrictions may occur due to components used to implement the IP. Examples include:
  - Vivado Debug Hub (BSCAN and BUFG)
  - IP modules with embedded global buffers or I/O
  - MIG controller (MMCM and BSCAN)
- Reconfigurable Modules must be initialized to ensure a predictable starting condition after reconfiguration. You can do this manually with a local reset, or with dedicated GSR events by selecting the `RESET_AFTER_RECONFIG` feature. `RESET_AFTER_RECONFIG` is always enabled for UltraScale devices.
- Decoupling logic is highly recommended to disconnect the reconfigurable region from the static portion of the design during the act of Partial Reconfiguration.
  - Clock and other inputs to Reconfigurable Modules can be decoupled to prevent spurious writes to memories during reconfiguration. This should be considered if `RESET_AFTER_RECONFIG` is not used.
  - The 2015.3 release introduced a new Partial Reconfiguration Decoupler IP. This IP allows users to easily insert MUXes to efficiently decouple AXI Lite, AXI4-Stream, and custom interfaces. More information on the PR Decoupler IP is available on the Xilinx website.
- A reconfigurable partition must be flooorplanned, so the module must be a block that can be contained by a Pblock and meet timing. If the module is complete, it is recommended to run this design through a non-PR flow to get an initial evaluation of placement, routing, and timing results. If the design has issues in a non-PR flow, these should be resolved before moving on to the PR flow.
- Each module pin on an RP has a partition pin. This is a routing point that connects static logic to the RP. If a design has too many partition pins for the number of available routing resources, routing congestion can occur. Consider the number of external pins on the RP, and develop a module that has a minimum required set of pins.
- Virtex-7 SSI devices (7V2000T, 7VX1140T, 7VH870T, 7VH580T) have two fundamental requirements. These requirements are:
  - Reconfigurable regions must be fully contained within a single SLR. This ensures that the global reset events are properly synchronized across all elements in the Reconfigurable Module, and that all super long lines (SLL) are contained within the static portion of the design. SLL are not partially reconfigurable.
  - If the initial configuration of a 7 series SSI device is done through an SPIx1 interface, partial bitstreams must be delivered to the ICAP located on the SLR where the Reconfigurable Partition exists, or to an external port, such as JTAG. If the initial configuration is done through any other configuration port, the master ICAP can be used as the delivery port for partial bitstreams.

- UltraScale devices have a new requirement related to partial reconfiguration events. Before a partial bitstream for a new Reconfigurable Module is loaded, the current Reconfigurable Module must be "cleared" to prepare for reconfiguration. For more information, see Summary of BIT Files for UltraScale Devices in Chapter 7.
- Dedicated encryption support for partial bitstreams is available natively. See Known Limitations, page 108 for specific unsupported use cases for UltraScale devices.
- Devices can use a per-frame CRC checking mechanism, enabled by `write_bitstream`, to ensure each frame is valid before loading.
- In situations where an RM flop is connected directly to a static I/O buffer, ensure there is no `IOB=TRUE` property on the RM flop. Failure to do so can lead to the register being illegally placed, as well as pre-`route_design` DRC errors.

Partial Reconfiguration is a powerful capability within Xilinx devices, and understanding the capabilities of the silicon and software is instrumental to success with this technology. While trade-offs must be recognized and considered during the development process, the overall result is a more flexible implementation of your FPGA design.

Partial Reconfiguration Licensing

Partial Reconfiguration is available as a licensed product within the Vivado Design Suite. Contact your local sales offices for pricing and ordering details.

A Partial Reconfiguration license is checked when the individual implementation commands (`opt_design`, `place_design`, `route_design`, `write_bitstream`) are called on designs containing the `HD.RECONFIGURABLE` property. A license is required to implement (place and route) a Partial Reconfiguration design and to generate partial bitstreams. No unique checks are done within the Vivado IDE, and the license feature is not held.

Common Applications

Overview

The basic premise of Partial Reconfiguration is that the device hardware resources can be time-multiplexed similar to the ability of a microprocessor to switch tasks. Because the device is switching tasks in hardware, it has the benefit of both flexibility of a software implementation and the performance of a hardware implementation. Several different scenarios are presented here to illustrate the power of this technology.

Networked Multiport Interface

Partial Reconfiguration optimizes traditional FPGA applications by reducing size, weight, power, and cost. Time-independent functions can be identified, isolated, and implemented as Reconfigurable Modules and swapped in and out of a single device as needed. A typical example is a 40G OTN muxponder application. The ports of the client side of the muxponder can support multiple interface protocols. However, it is not possible for the system to predict which protocol will be used before the FPGA is configured. To ensure that the FPGA does not have to be reconfigured and thus disable all ports, every possible interface protocol is implemented for every port, as illustrated in Figure 2-1, page 18.

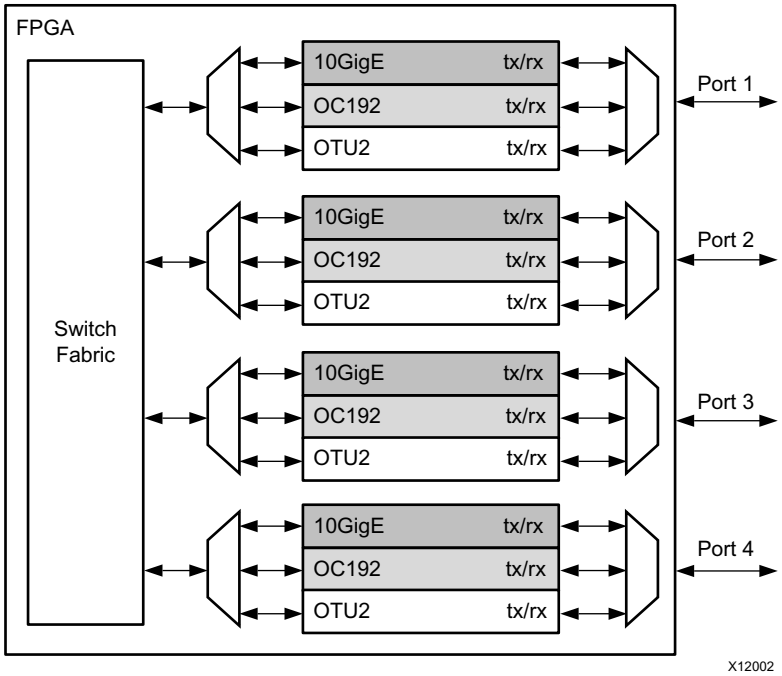

Figure 2-1: **Network Switch Without Partial Reconfiguration**

This is an inefficient design because only one of the standards for each port is in use at any point in time. Partial Reconfiguration enables a more efficient design by making each of the port interfaces a Reconfigurable Module, as shown in Figure 2-2. This also eliminates the MUX elements required to connect multiple protocol engines to one port.

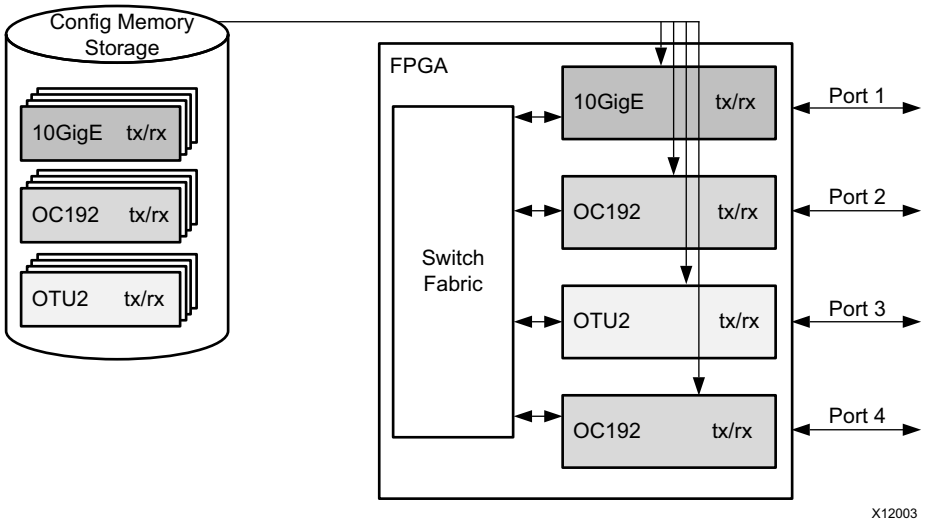

Figure 2-2: **Network Switch With Partial Reconfiguration**

A wide variety of designs can benefit from this basic premise. Software defined radio (SDR), for example, is one of many applications that has mutually exclusive functionality, and which sees a dramatic improvement in flexibility and resource usage when this functionality is multiplexed.

There are additional advantages with a partially reconfigurable design other than efficiency. In the Figure 2-2, page 18 example, a new protocol can be supported at any time without affecting the static logic, the switch fabric in this example. When a new standard is loaded for any port, the other existing ports are not affected in any way. Additional standards can be created and added to the configuration memory library without requiring a complete redesign. This allows greater flexibility and reliability with less down time for the switch fabric and the ports. A debug module could be created so that if a port was experiencing errors, an unused port could be loaded with analysis/correction logic to handle the problem real-time.

In the Figure 2-2, page 18 example, a unique partial BIT file must be generated for each unique physical location that could be targeted by each protocol. Partial BIT files are associated with an explicit region on the device. In this example, sixteen unique partial BIT files to accommodate four protocols for four locations.

Configuration by Means of Standard Bus Interface

Partial Reconfiguration can create a new configuration port using an interface standard more compatible with the system architecture. For example, the FPGA could be a peripheral on a PCIe® bus and the system host could configure the FPGA through the PCIe connection. After power-on reset the FPGA must be configured with a full BIT file. However, the full BIT file might only contain the PCIe interface and connection to the internal configuration access port (ICAP).

Bitstream compression can be used to reduce the size and therefore configuration time of this initial device load, helping the FPGA configuration meet PCIe enumeration specifications.

The system host could then configure the majority of the FPGA functionality with a partial BIT file downloaded through the PCIe port as shown in Figure 2-3.

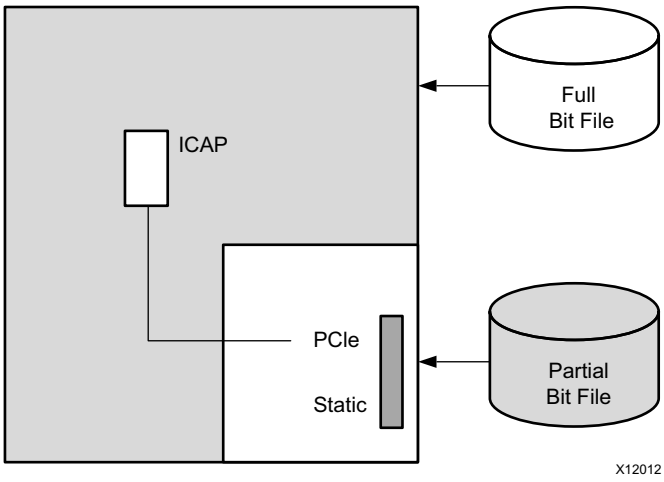

Figure 2-3: **Configuration by Means of PCIe Interface**

The PCIe standard requires the peripheral (the FPGA in this case) to acknowledge any requests even if it cannot service the request. Reconfiguring the entire FPGA would violate this requirement. Because the PCIe interface is part of the static logic, it is always active during the Partial Reconfiguration process, thus ensuring that the FPGA can respond to PCIe commands even during reconfiguration.

Tandem Configuration is a related solution that at first glance appears to be the same as is shown here. However, the solution using Partial Reconfiguration differs from Tandem Configuration on 7 series devices in two regards:

- First, the configuration process with PR is a full device configuration, made smaller and faster through compression, followed by a partial bitstream that overwrites the black box region to complete the overall configuration. Tandem Configuration is a two-stage configuration where each configuration frame is programmed exactly once.
- Second, Tandem Configuration for 7 series devices does not permit dynamic reconfiguration of the user application. Using Partial Reconfiguration, the dynamic region can be reloaded with different user applications or field updates.

Tandem Configuration is designed to be a specific solution for a specific goal: fast configuration of a PCIe endpoint to meet enumeration requirements. For more information, see the following manuals:

- *7 Series FPGAs Integrated Block for PCI Express Product Guide* (PG054) [Ref 13]
- *Virtex-7 FPGA Gen3* PCIe *Integrated Block for PCI Express Product Guide* (PG023) [Ref 14]
- *LogiCORE IP UltraScale FPGAs Gen3 Integrated Block for PCI Express Product Guide* (PG156) [Ref 15]

Dynamically Reconfigurable Packet Processor

A packet processor can use Partial Reconfiguration to change its processing functions quickly, based on the packet types received. In Figure 2-4, a packet has a header that contains the partial BIT file, or a special packet contains the partial BIT file. After the partial BIT file is processed, it is used to reconfigure a coprocessor in the FPGA. This is an example of the FPGA reconfiguring itself based on the data packet received instead of relying on a predefined library of partial BIT files.

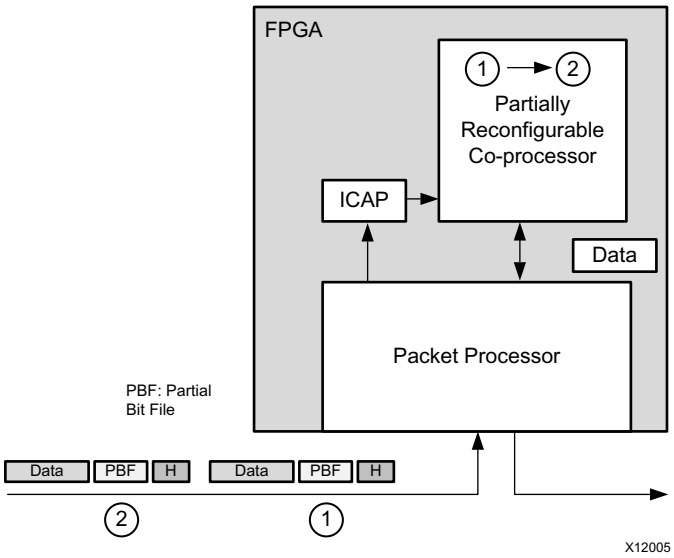

Figure 2-4: **Dynamically Reconfigurable Packet Processor**

Asymmetric Key Encryption

There are some new applications that are not possible without Partial Reconfiguration. A very secure method for protecting the FPGA configuration file can be architected when Partial Reconfiguration and asymmetric cryptography are combined. (See Public-key cryptography for asymmetric cryptography details.)

In Figure 2-5, the group of functions in the shaded box can be implemented within the physical package of the FPGA. The cleartext information and the private key never leave a well-protected container.

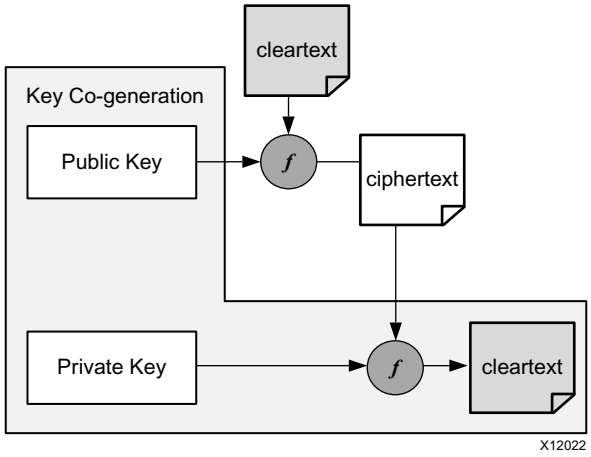

Figure 2-5: **Asymmetric Key Encryption**

In a real implementation of this design, the initial BIT file is an unencrypted design that does not contain any proprietary information. The initial design only contains the algorithm to generate the public-private key pair and the interface connections between the host, FPGA and ICAP.

After the initial BIT file is loaded, the FPGA generates the public-private key pair. The public key is sent to the host which uses it to encrypt a partial BIT file. The encrypted partial BIT file is downloaded to the FPGA where it is decrypted and sent to the ICAP to partially reconfigure the FPGA, as shown in Figure 2-6, page 23.

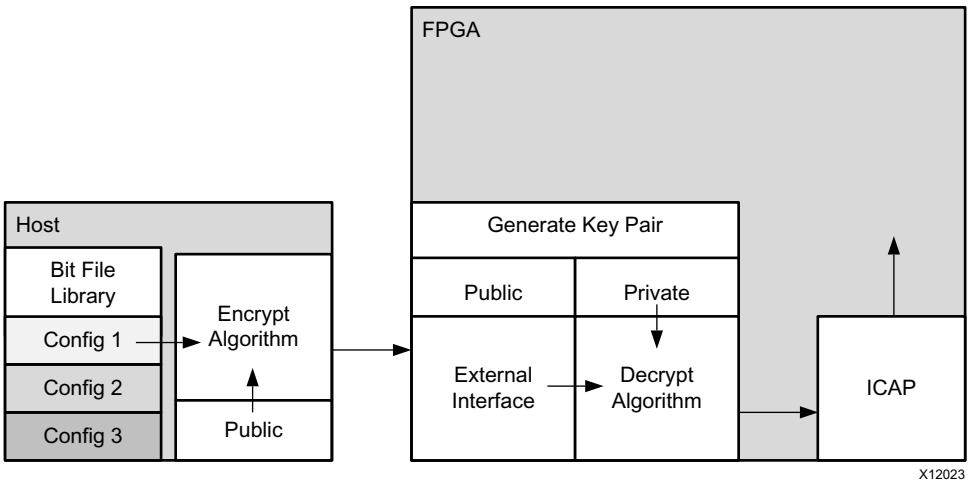

Figure 2-6: **Loading an Encrypted Partial Bit File**

The partial BIT file could be the vast majority of the FPGA design with the logic in the static design consuming a very small percentage of the overall FPGA resources.

This scheme has several advantages:

- The public-private key pair can be regenerated at any time. If a new configuration is downloaded from the host it can be encrypted with a different public key. If the FPGA is configured with the same partial BIT file, such as after a power-on reset, a different public key pair is used even though it is the same BIT file.
- The private key is stored in SRAM. If the FPGA ever loses power the private key no longer exists.
- Even if the system is stolen and the FPGA remains powered, it is extremely difficult to find the private key because it is stored in the general purpose FPGA programmable logic. It is not stored in a special register. You could manually locate each register bit that stores the private key in physically remote and unrelated regions.

Summary

In addition to reducing size, weight, power and cost, Partial Reconfiguration enables new types of FPGA designs that would otherwise be impossible to implement.

Vivado Software Flow

Overview

The Vivado® Partial Reconfiguration design flow is similar to a standard design flow, with some notable differences. The implementation software automatically manages the low-level details to meet silicon requirements. You must provide guidance to define the design structure and floorplan. The following steps summarize processing a PR design:

1. Synthesize the static and Reconfigurable Modules separately.
2. Create physical constraints (Pblocks) to define the reconfigurable regions.
3. Set the `HD.RECONFIGURABLE` property on each Reconfigurable Partition.
4. Implement a complete design (static and one Reconfigurable Module per Reconfigurable Partition) in context.
5. Save a design checkpoint for the full routed design.
6. Remove Reconfigurable Modules from this design and save a static-only design checkpoint.
7. Lock the static placement and routing.
8. Add new Reconfigurable Modules to the static design and implement this new configuration, saving a checkpoint for the full routed design.
9. Repeat Step 8 until all Reconfigurable Modules are implemented.
10. Run a verification utility (`pr_verify`) on all configurations.
11. Create bitstreams for each configuration.

Partial Reconfiguration Commands

The PR flows are currently only supported through the non-project batch/Tcl interface (no project based commands). Example scripts are provided in the *Vivado Design Suite Tutorial: Partial Reconfiguration* (UG947) [Ref 1], along with step-by-step instructions for setting up the flows. See that Tutorial for more information.

The following sections describe a few specialized commands and options needed for the PR flows. Examples of how to use these commands to run a PR flow are given. For more information on individual commands, see the *Vivado Design Suite Tcl Command Reference Guide* (UG835) [Ref 16].

Synthesis

Synthesizing a partially reconfigurable design does not require any special commands, but does require bottom-up synthesis. There are currently no unsupported commands for synthesis, optimization, or implementation.

These synthesis tools are supported:

- XST (supported for 7 series only)
- Synplify
- Vivado Synthesis

IMPORTANT: *NGC format files are not supported in the Vivado Design Suite for UltraScale devices. Xilinx recommends that you regenerate the IP using the Vivado Design Suite IP customization tools with native output products. Alternatively, you can use the* `NGC2EDIF` *command to migrate the NGC file to EDIF format for importing. However, Xilinx recommends using native Vivado IP rather than XST-generated NGC format files going forward.*

IMPORTANT: *Bottom-up synthesis refers to a synthesis flow in which each module has its own synthesis project. This generally involves turning off automatic I/O buffer insertion for the lower level modules.*

This document only covers the Vivado synthesis flow.

Synthesizing the Top Level

You must have a top-level netlist with a black box for each Reconfigurable Partition (RP). This requires the top-level synthesis to have module or entity declarations for the partitioned instances, but no logic; the module is empty.

The top-level synthesis infers or instantiates I/O buffers on all top level ports. For more information on controlling buffer insertion, see this link in the *Vivado Design Suite User Guide: Synthesis* (UG901) [Ref 17].

```
synth_design -flatten_hierarchy rebuilt -top <top_module_name> -part <part>
```

Synthesizing Reconfigurable Modules

Because each Reconfigurable Module must be instantiated in the same black box in the static design, the different versions must have identical interfaces. The name of the block must be the same in each instance, and all the properties of the interfaces (names, widths, direction) must also be identical. Each configuration of the design is assembled like a flat design.

To synthesize a Reconfigurable Module, turn off all buffer insertions. You can do so in Vivado Synthesis using the `synth_design` command in conjunction with the `-mode out_of_context` switch:

```
synth_design -mode out_of_context -flatten_hierarchy rebuilt -top
<reconfig_module_name> -part <part>
```

Table 3-1: **`synth_design` Options**

Command Option	Description
`-mode out_of_context`	Prevents I/O insertion for synthesis and downstream tools. The `out_of_context` mode is saved in the checkpoint if `write_checkpoint` is issued.
`-flatten_hierarchy rebuilt`	There are several values allowed for `-flatten_hierarchy`, but `rebuilt` is the recommended setting for PR flows.
`-top`	This is the module/entity name of the module being synthesized.
`-part`	This is the Xilinx® part being targeted (for example, `xc7k325tffg900-3`)

The `synth_design` command synthesizes the design and stores the results in memory. In order to write the results out to a file, use:

```
write_checkpoint <file_name>.dcp
```

It is recommended to close the design in memory after synthesis, and run implementation separately from synthesis.

Reading Design Modules

If there is currently no design in memory, you must load a design. This can be done in a variety of ways, for either the static design or for Reconfigurable Modules. After the configurations are implemented, checkpoints are exclusively used to read in placed and routed module databases.

Method 1: Read Netlist Design

This approach should be used when modules have been synthesized by tools other than Vivado synthesis.

```
read_edif <top>.edf/edn/ngc
read_edif <rp1_a>.edf/edn/ngc
read_edif <rp2_a>.edf/edn/ngc
link_design -top <top_module_name> -part <part>
```

Table 3-2: **`link_design` Options**

Command Option	Description
`-part`	This is the Xilinx part being targeted (for example, `xc7k325tffg900-3`)
`-top`	This is the module/entity name of the module being implemented. This switch can be omitted if `set_property top <top_module_name> [current_fileset]` is issued prior to `link_design`.

Method 2: Open/Read Checkpoint

If the static (top-level) design has synthesis or implementation results stored as a checkpoint, then it can be loaded using the `open_checkpoint` command. This command reads in the static design checkpoint and opens it in active memory:

```
open_checkpoint <file>
```

If the checkpoint is for a reconfigurable module (that is, not for static), then the instance name must be specified using `read_checkpoint -cell`. If the checkpoint is a post-implementation checkpoint, then the additional `-strict` option must be used as well. This option can also be used with a post-synthesis checkpoint to ensure exact port matching has been achieved. To read in a checkpoint in a Reconfigurable Module, the top-level design must already be opened, and must have a black box for the specified cell. Then the following command can be specified:

```
read_checkpoint -cell <cellname > <file> [-strict]
```

Table 3-3: **`read_checkpoint` Switches**

Switch Name	Description
`-cell`	Specifies the full hierarchical name of the Reconfigurable Module.
`-strict`	Requires exact ports match for replacing cell, and checks that part, package, and speed grade values are identical. Should be used when restoring implementation data.
`<file>`	Specifies the full or relative path to the checkpoint (DCP) to be read in.

Method 3: Open Checkpoint/Update Design

This is useful when the synthesis results are in the form of a netlist (EDF or EDN), but static has already been implemented. The following example shows the commands for the second configuration in which this is true.

```
open_checkpoint <top>.dcp
lock_design -level routing
update_design -cells <rp1> -from_file <rp1_b>.{edf/edn}
update_design -cells <rp2> -from_file <rp2_b>.{edf/edn}
```

Adding Reconfigurable Modules with Sub-Module Netlists

If a Reconfigurable Module has sub-module netlists, it can be difficult for the Vivado tools to process the sub-module netlists. This is because in the PR flow the RM netlists are added to a design that is already open in memory. This means the `update_design -cells` command must be used, which requires the cell name for every EDIF file, which can be troublesome to get.

There are two ways to make loading RM sub-module netlists easier in the Vivado Design Suite.

Method 1: Create a Single RM Checkpoint (DCP)

Create an RM checkpoint (DCP) that includes all netlists. Use `add_files` to add all of the EDIF (or NGC) files, and use `link_design` to resolve the EDIF files to their respective cells. Here is an example of the commands used in this process:

```
add_files [list rm.edf ip_1.edf … ip_n.edf]

# Run if RM XDC exists
add_files rm.xdc

link_design -top <rm_module> -part <part>
write_checkpoint rm_v#.dcp
close_project
```

IMPORTANT: *Using this methodology to combine/convert a netlist into a DCP is the recommended way to handle an RM that has one or more NGC source files as well.*

Then this newly-created RM checkpoint can be used in the PR flow. In the commands below, the single `read_checkpoint -cell` command replaces what could be many `update_design -cell` commands.

```
add_files static.dcp
link_design -top <top> part <part>
lock_design -level routing
read_checkpoint -cell <rm_inst> rm_v#.dcp
```

Method 2: Place the Sub-Module Netlists in the Same Directory as the RM's Top-Level Netlist

When the top-level RM netlist is read into the PR design using `update_design -cell`, make sure that all sub-module netlists are in the same directory as the RM top-level netlist. In this case, the lower-level netlists do not need to be specified, but they are picked up automatically by the `update_design -cells` command. This is less explicit than Method 1, but requires fewer steps. In this case the commands to load the RM netlist would look like the following:

```
add_files static.dcp
link_design -top <top> part <part>
lock_design -level routing
update_design -cells <rm_inst> -from_file rm_v#.edf
```

In the last (`update_design`) command above, the lower-level netlists are picked up automatically if they are in the same directory as `rm_v#.edf`.

Reading Design Constraints

New constraints can be applied for each configuration at various points in the flow. If an RM is read in as a DCP, then any constraints stored in the DCP are automatically applied. Additionally, the `read_xdc` command can be used to apply constraints scoped to the top-level, or to the specific cell (using `-cell` switch). If constraint are expected to directly or indirectly affect the RM, then the RM must be resolved (not a black box) prior to reading in the new constraints. Otherwise, the constraints may be dropped or not correctly propagated in the constraint system. Because Static is only placed and routed in the initial configuration, all constraints for subsequent configurations (where Static is locked) should be focused strictly on the RP regions being implemented.

Implementation

Because the PR flow allows for various configurations in hardware, multiple implementation runs are required. Each implementation of a PR design is referred to as a *configuration*. Each module of the design (static or Reconfigurable Module) can be implemented or imported (if previously implemented). Implementation results for the static design must be consistent for each configuration, so that the design is implemented in one configuration, and then imported in subsequent configurations. Additional configurations can be constructed by importing static, and implementing or importing each Reconfigurable Module.

There are no restrictions to the support of implementation commands or options for PR, but certain optimizations and sub-routines are not done if they oppose the fundamental requirements of partial reconfiguration. The following list of commands can be run after the logical design is loaded (using `link_design` or `open_checkpoint`):

```
# Run if all constraints are not already loaded
read_xdc

# Optional command
opt_design

place_design

# Optional command
phys_opt_design

route_design
```

Preserving Implementation Data

In the PR flow, it is a requirement to lock down the placement and routing results of the static logic from the first configuration for all subsequent configurations. The static implementation of the first configuration must be saved as a checkpoint. When the checkpoint is read for subsequent configurations, the placement and routing must be locked, to ensure that the static design remains completely identical from configuration to configuration. To lock the placement and routing of an imported checkpoint (static or reconfigurable), the `lock_design` command is used.

```
lock_design -level [logical|placement|routing] [cell_name]
```

When locking down the static logic with the above command, the optional `[cell_name]` can be omitted.

```
lock_design -level routing
```

To lock the results of an imported RM, the full hierarchical name should be specified within the post-implementation checkpoint:

```
lock_design -level routing u0_RM_instance
```

For Partial Reconfiguration, the only supported preservation level is `routing`. Other preservation levels are available for this command, but they must only be used for other Hierarchical Design flows.

Partial Reconfiguration Constraints and Properties

There are properties and constraints unique to the Partial Reconfiguration flow. These initiate PR-specific implementation processing and apply specific characteristics in the partial bitstreams. The four areas for constraints and properties for partial reconfiguration are:

Constraints and Properties	Necessity
Defining a module as reconfigurable	Required
Creating a floorplan for the reconfigurable region	Required
Applying reset after reconfiguration	Optional, but highly recommended
Turn on visualization scripts	Optional

Define a Module as Reconfigurable

In order to implement a PR design, it is required to specify each Reconfigurable Module as such. To do this you must set a property on the top level of each hierarchical cell that is going to be reconfigurable. For example, take a design where one Reconfigurable Partition named `inst_count` exists, and it has two Reconfigurable Modules, `count_up` and `count_down`. The following command must be issued prior to implementation of the first configuration.

```
set_property HD.RECONFIGURABLE TRUE [get_cells inst_count]
```

This initiates the Partial Reconfiguration features in the software that are required to successfully implement a PR design. The `HD.RECONFIGURABLE` property implies a number of underlying constraints and tasks:

- Sets `DONT_TOUCH` on the specified cell and its interface nets. This prevents optimization across the boundary of the module.
- Sets `EXCLUDE_PLACEMENT` on the cell's Pblock. This prevents static logic from being placed in the reconfigurable region.
- Sets `CONTAIN_ROUTING` on the cell's Pblock. This keeps all the routing for the Reconfigurable Module within the bounding box.
- Enables special code for DRCs, clock routing, etc.

Create a Floorplan for the Reconfigurable Region

Each Reconfigurable Partition is required to have a Pblock to define the physical resources available for the Reconfigurable Module. Because this Pblock is set on a Reconfigurable Partition, these restrictions and requirements apply:

- The Pblock must contain only valid reconfigurable element types. The region may overlap other site types, but these other sites must not be included in the `resize_pblock` commands.
- Multiple Pblock rectangles for each component type may be used to create the Reconfigurable Partition region, but for the greatest routability, they should be contiguous. Gaps to account for non-reconfigurable resources are permitted, but in general, the simpler the overall shape, the easier the design will be to place and route.
- If using the `RESET_AFTER_RECONFIG` property for 7 series devices, the Pblock height must align to clock region boundaries. See Apply Reset After Reconfiguration for more detail.
- The width and composition of the Pblock must not split interconnect columns for 7 series devices. See Creating Pblocks for 7 Series Devices in Chapter 5 for more detail.
- The largest RM needs to be taken into consideration when defining the Pblock in SVD parts. Otherwise, the design can be overutilized and `write_bitstream` will generate an error.
- The Pblock must not overlap any other Pblock in the design.
- Nesting of Reconfigurable Partitions (a Reconfigurable Partition within another Reconfigurable Partition) is currently not supported. Standard pblocks for floorplanning logic within a Reconfigurable Partition are supported, as are nested Pblocks.

Table 3-4: **Pblock Commands and Properties**

Command/Property Name	Description
`create_pblock`	Command used to create the initial Pblock for each Reconfigurable Partition instance.
`add_cells_to_pblock`	Command used to specify the instances that belong to the Pblock. This is typically a level of hierarchy as defined by the bottom-up synthesis processing.
`resize_pblock`	Command used to define the site types (such as SLICE or RAMB36) and site locations that are owned by the Pblock.
`RESET_AFTER_RECONFIG`	Pblock property used to control the use of the dedicated GSR event on the reconfigurable region. Use of this property is highly recommended and, for 7 series and Zynq devices, requires clock region alignment in the vertical direction.
`CONTAIN_ROUTING`	Pblock property used to control the routing to prevent usage of routing resources not owned by the Pblock. This property is mandatory for PR and is set to True automatically for Reconfigurable Partitions. Static routing is still allowed to use resources inside of the Pblock.

Table 3-4: **Pblock Commands and Properties**

Command/Property Name	Description
EXCLUDE_PLACEMENT	Pblock Property used to prevent the placement of any logic, not belonging to the Pblock, inside the defined Pblock RANGE. This property is mandatory for PR and set to true automatically for Reconfigurable Partitions. Static logic can be placed inside of the Reconfigurable Partition with a specific LOC property if RESET_AFTER_RECONFIG is not used.
PARTPIN_SPREADING	Used to control the maximum number of PartPins per INT tile. Default is 5. Setting a lower value (i.e. 3) increases the spreading between partition pin placements. This typically eases routing congestion in areas with dense PartPin placement, but can negatively affect RP interface timing.

The following is an example of a set of constraints for a Reconfigurable Partition:

```
#define a new pblock
create_pblock pblock_count

#add a hierarchical module to the pblock
add_cells_to_pblock [get_pblocks pblock_count] [get_cells [list inst_count]]

#define the size and components within the pblock
resize_pblock [get_pblocks pblock_count] -add {SLICE_X136Y50:SLICE_X145Y99}
resize_pblock [get_pblocks pblock_count] -add {RAMB18_X6Y20:RAMB18_X6Y39}
resize_pblock [get_pblocks pblock_count] -add {RAMB36_X6Y10:RAMB36_X6Y19}
```

Floorplan in the Vivado IDE

Although Project Mode support is not available, the Vivado IDE can be used for planning and visualization tasks. The best example of this is using the Device view to create and modify Pblock constraints for floorplanning.

1. Open the synthesized static design and the largest of each Reconfigurable Module. Here are the commands, using the tutorial design (found in the *Vivado Design Suite Tutorial: Partial Reconfiguration* (UG947) [Ref 1]) as an example:

   ```
   open_checkpoint synth/Static/top_synth.dcp
   read_checkpoint -cell [get_cells inst_count] synth/count_up/count_synth.dcp
   read_checkpoint -cell [get_cells inst_shift] synth/shift_right/shift_synth.dcp
   set_property HD.RECONFIGURABLE true [get_cells inst_count]
   set_property HD.RECONFIGURABLE true [get_cells inst_shift]
   ```

   At this point, a full configuration has been loaded into memory, and the Reconfigurable Partitions have been defined.

2. To create Pblock constraints for the Reconfigurable Partitions, right-click on an instance in the Netlist window (in this case, inst_count or inst_shift) and select **Draw Pblock**. Create a rectangle in the Device view to select resources for this Reconfigurable Partition.
3. With this Pblock selected, note that the Pblock Properties pane shows the number of available and required resources. The number required is based on the currently loaded Reconfigurable Module, so keep in mind that other modules may have different requirements. If additional rectangles are required to build the appropriate shape (an "L", for example), right-click the Pblock in the Device view and select **Add Pblock Rectangle**.
4. Design rule checks (DRCs) can be issued to validate the floorplan and other design considerations for the in-memory configuration. To run, select **Tools > Report > Report DRC** and ensure the Partial Reconfiguration checks are present (see Figure 3-1, page 35). Note that if `HD.RECONFIGURABLE` has not been set on a Pblock, only a single DRC is available, instead of the full complement shown below.

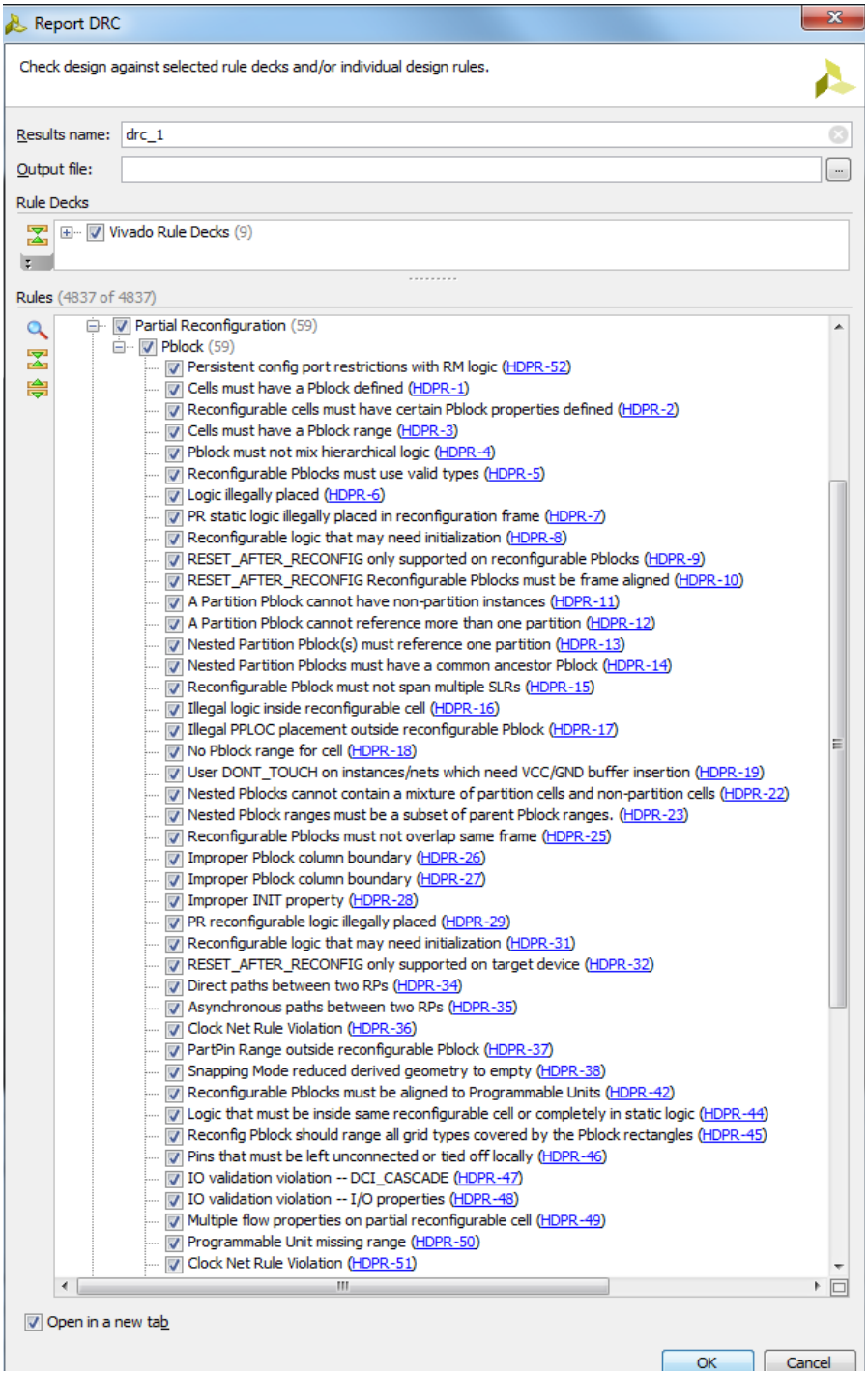

Figure 3-1: **Partial Reconfiguration DRCs in the Vivado IDE**

This set of DRCs can be run from the Tcl Console or within a script, by using the `report_drc` command. To limit the checks to the ones shown here for Partial Reconfiguration, use this syntax:

```
report_drc -checks [get_drc_checks HDPR*]
```

To extend the DRCs to those checked during specific phases of design processing the `-ruledeck` option can be used. For example, the following command can be issued on a placed and routed design:

```
report_drc -ruledeck bitstream_checks
```

To save these floorplanning constraints, enter the following command in the Tcl Console:

```
write_xdc top_fplan.xdc
```

The Pblock constraints stored in this constraints file can be used directly or can be copied to another top-level design constraints file. This XDC file contains all the constraints in the current design in memory not just the constraints recently added.

CAUTION! *Do NOT save the overall design from the Vivado IDE using **File > Save Checkpoint** or the equivalent button. If you save the currently loaded design in this way, you will overwrite your synthesized static design checkpoint with a new version that includes Reconfigurable Modules and additional constraints.*

Timing Constraints

Timing constraints for a Partial Reconfigurable design are similar to timing constraints for a traditional flat design. The primary clocks and I/Os must be constrained with the corresponding constraints. For more information on these constraints, see this link (for defining clocks) and this link (for constraining I/O delays) in the *Vivado Design Suite User Guide: Using Constraints* (UG903) [Ref 18].

After the correct constraints are applied to the design, run static timing analysis to verify the performance of the design. This verification must be run for each reconfigurable module in the overall static design. For more information on how to analyze the design, see the *Vivado Design Suite User Guide: Design Analysis and Closure Techniques* (UG906) [Ref 19].

The Vivado Design Suite includes the capability to run cell level timing reports. Use the `-cell` option for `report_timing` or `report_timing_summary` to focus timing analysis on a specific Reconfigurable Module. This is especially useful on configurations where the static design has been imported and locked from a prior configuration.

The current constraint set does not allow the use of constraining or timing analysis for interconnect tiles at the reconfigurable module boundary. The ability to constrain and analyze for the interconnect tiles is being researched for a future release.

Partition Pins

Interface points called partition pins are automatically created within the Pblock ranges defined for the Reconfigurable Partition. These virtual I/O are established within interconnect tiles as the anchor points that remain consistent from one module to the next. No physical resources such as LUTs or flip-flops are required to establish these anchor points, and no additional delay is incurred at these points.

The placer chooses locations based on source and loads and timing requirements, but you can specify these locations as well. The following constraints can be applied to influence partition pin placement.

Table 3-5: **Context Properties**

Command/Property Name	Description
`HD.PARTPIN_LOCS`	Used to define a specific interconnect tile (INT) for the specified port to be routed. Overrides an `HD.PARTPIN_RANGE` value. Affects placement and routing of logic on both sides of the Reconfigurable Partition boundary. Do not use this property on clock ports, as this assumes local routing for the clock. Do not use this property on dedicated connections.
`HD.PARTPIN_RANGE`	Used to define a range of component sites (SLICE, DSP, BRAM) or interconnect tiles (INT) that can be used to route the specified port(s). The value is automatically calculated based on Pblock range if no user-defined `HD.PARTPIN_RANGE` value exists.

Note: The `PARTPIN_SPREADING` command in Table 3-4, page 32, can also be used to affect Partition Pins, but is applied at the Pblock level.

Context Property Examples:

- `set_property HD.PARTPIN_LOCS INT_R_X4Y153 [get_ports <port_name>]`
- `set_property HD.PARTPIN_RANGE SLICE_X4Y153:SLICE_X5Y157 [get_ports <port_name>]`

Instance names for interconnect tile sites can be seen in the Device View with the Routing Resources enabled.

Note: The `HD.PARTPIN_RANGE` is automatically set during `place_design` if no user-defined value is found. Once the value is set, it will not be reset during interactive place and route, such as making experimental changes to the RP Pblocks and running `place_design -unplace`. In this case, the `HD.PARTPIN_RANGE` and `HD.PARTPIN_LOCS` need to be reset manually if Pblock adjustments are made. The properties can be reset like most properties.

The following Tcl proc can be useful when doing this kind of interactive floorplanning on PR designs:

```
################################################
Proc to unroute, uplace, and reset HD.PARTPIN_*
################################################
proc pr_unplace {} {
  route_design -unroute
  place_design -unplace
  set cells [get_cells -quiet -hier -filter HD.RECONFIGURABLE]
  foreach cell $cells {
    reset_property HD.PARTPIN_LOCS [get_pins $cell/*]
    reset_property HD.PARTPIN_RANGE [get_pins $cell/*]
  }
}
```

Apply Reset After Reconfiguration

With the Reset After Reconfiguration feature, the reconfiguring region is held in a steady state during partial reconfiguration, and then all logic in the new Reconfigurable Module is initialized to its starting values. Static routes can still freely pass unaffected through the region, and static logic (and all other PR regions) elsewhere in the device continue to operate normally during Partial Reconfiguration. Partial Reconfiguration with this feature behaves in the same manner as the initial configuration of the FPGA, with synchronous elements being released in a known, initialized state.

IMPORTANT: *Release of global signals such as GSR (Global Set Reset) and GWE (Global Write Enable) are not guaranteed to be synchronized chip-wide. If functionality within a Reconfigurable Module relies on synchronized startup of initialized sequential elements, the clock(s) driving the logic in that module or Clock Enables on these elements can be disabled during reconfiguration, then re-enabled after reconfiguration has been completed. For more details, see the "Design Advisory for techniques on properly synchronizing flip-flops and SRLs" answer record (AR#44174) [Ref 31].*

This is the `RESET_AFTER_RECONFIG` property syntax:

```
set_property RESET_AFTER_RECONFIG true [get_pblocks <reconfig_pblock_name>]
```

If the design uses the DRP interface of the 7 series XADC component, the interface will be blocked (held in reset) during partial reconfiguration when RESET_AFTER_RECONFIG is enabled. The interface will be non-responsive (busy), and there will be no access during the length of the partial reconfiguration period. The interface will become accessible again after partial reconfiguration is complete.

To apply the Reset After Reconfiguration methodology for 7 series and Zynq-7000 AP SoC devices, Pblock constraints must align to reconfigurable frames. Because the GSR affects every synchronous element within the region, exclusive use of reconfiguration frames is required; static logic is not permitted within these reconfigurable frames (static routing is permitted). Pblocks must align vertically to clock regions, since that matches the base region for a reconfigurable frame. The width of a Pblock does not matter when using `RESET_AFTER_RECONFIG`.

UltraScale™ devices do not have this clock region alignment requirement, and GSR can be applied at a fine granularity. Because of this, `RESET_AFTER_RECONFIG` is automatically applied for all Reconfigurable Partitions in the UltraScale architecture.

In Figure 3-2, the Pblock on the left (`pblock_shift`) is frame-aligned because the top and bottom of the Pblock align to the height of clock region X1Y3. The Pblock on the right (`pblock_count`) is not frame-aligned.

- For 7 series devices: Pblocks that are not frame-aligned (such as `pblock_count` in the figure below) cannot have `RESET_AFTER_RECONFIG` set because any static logic placed between it and the clock region boundary above it would be affected by GSR after that module was partially reconfigured.
- For UltraScale devices: because of the improved GSR controls, both Pblocks automatically use `RESET_AFTER_RECONFIG`.

Using the `SNAPPING_MODE` constraint automatically creates legal, reconfigurable Pblocks. See Automatic Adjustments for Reconfigurable Partition Pblocks in Chapter 5 (for 7 series devices) or Automatic Adjustments for PU on PBlocks in Chapter 6 (for UltraScale devices) for more information.

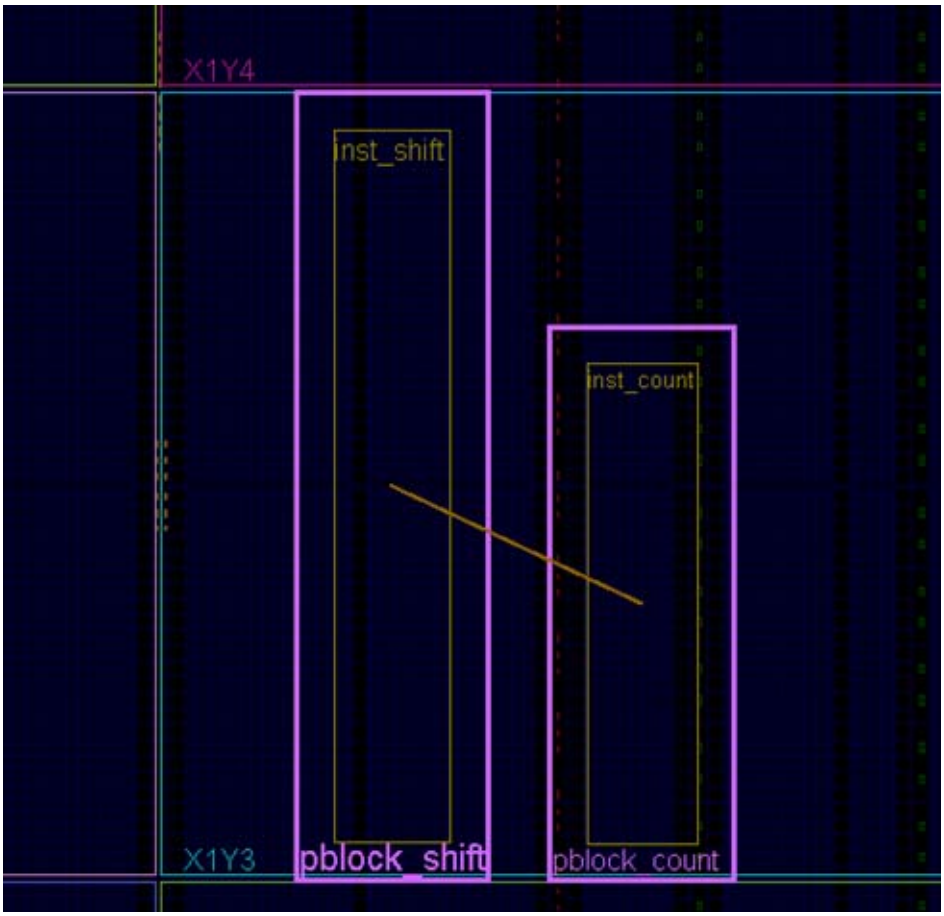

Figure 3-2: **RESET_AFTER_RECONFIG Compatible (Left) and Incompatible (Right) Pblocks**

The GSR capabilities are embedded within the partial bitstreams, so nothing extra must be done to include this feature during reconfiguration. However, because this process utilizes the SHUTDOWN sequence (masked to the reconfiguring region only), the external DONE pin are pulled LOW when reconfiguration starts, then pull HIGH when it successfully completes. This behavior must be considered when setting up the board. Using the STARTUP block DONEO is not an option to prevent the DONE pin from changing state, since this block is disabled during shutdown. Nor can STARTUP be used for other purposes, such as generating a configuration clock for partial reconfiguration if RESET_AFTER_RECONFIG is used.

An alternative approach would be to forego this property and apply a local reset to any reconfigured logic that requires initialization to function properly. This approach does not require vertical alignment to clock region boundaries. Without GSR or a local reset, the initial starting value of a synchronous element within a reconfigured module cannot be guaranteed.

Turn On Visualization Scripts

The configuration tiles that are part of the partial bitstreams can be visualized within the Device View in the Vivado IDE. These are identified by scripts that are created during implementation. To turn on script creation, set this parameter before starting implementation:

```
set_param HD.VISUAL true
```

This generates multiple scripts placed in the `hd_visual` directory, which is created in the directory where the run script is launched. To use these scripts, read a routed design checkpoint into the Vivado IDE, then source one of the scripts. These design-specific scripts highlight configuration tiles as you have defined them, show configuration frames used to create the partial bit file, or show sites excluded by the PR floorplan. Additional scripts are created for other flows, such as Module Analysis or Tandem Configuration, and are not used for PR.

Software Flow

This section describes the basic flow, and gives sample commands to execute this flow.

Synthesis

Each module (including Static) needs to be synthesized bottom-up so that a netlist or checkpoint exists for static and each Reconfigurable Module.

1. Synthesize the top level:

   `read_verilog top.v` (and other HDL associated with the static design, including black box module definitions for Reconfigurable Modules)

   then:

   ```
   read_xdc top_synth.xdc
   synth_design -top top -part xc7k70tfbg676-2
   write_checkpoint top_synth.dcp
   ```

2. Synthesize a Reconfigurable Module:

```
read_verilog rp1_a.v
synth_design -top rp1 -part xc7k70tfbg676-2 -mode out_of_context
write_checkpoint rp1_a_synth.dcp
```

3. Repeat for each remaining Reconfigurable Module

```
read_verilog rp1_b.v
synth_design -top rp1 -part xc7k70tfbg676-2 -mode out_of_context
write_checkpoint rp1_b_synth.dcp
```

Implementation

Create as many configurations as necessary to implement all Reconfigurable Modules at least once. The first configuration loads in synthesis results for top and the first Reconfigurable Module. You must then mark the module as being reconfigurable, then run implementation. Write out a checkpoint for the complete routed configuration, and optionally for the Reconfigurable Module so it can be reused later if desired. Finally, remove the Reconfigurable Module from the design (`update_design -cell -black_box`) and write out a checkpoint for the locked static design alone.

Configuration 1:

```
open_checkpoint top_synth.dcp
read_xdc top_impl.xdc
read_checkpoint -cell rp1 rp1_a_synth.dcp
set_property HD.RECONFIGURABLE true [get_cells rp1]
opt_design
place_design
route_design
write_checkpoint config1_routed.dcp
write_checkpoint -cell rp1 rp1_a_route_design.dcp
update_design -cell rp1 -black_box
lock_design -level routing
write_checkpoint static_routed.dcp
```

For the second configuration, load the placed and routed checkpoint for static (if it was closed), which currently has a black box for the Reconfigurable Module. Then load in the synthesis results for the second Reconfigurable Module and implement the design. Finally write out an implementation checkpoint for the second version of the Reconfigurable Module.

Configuration 2:

```
open_checkpoint static_routed.dcp
read_checkpoint -cell rp1 rp1_b_synth.dcp
opt_design
place_design
route_design
write_checkpoint config2_routed.dcp
write_checkpoint -cell rp1 rp1_b_route_design.dcp
```

TIP: *Keep each configuration in a separate folder so that all intermediate checkpoints, log and report files, bit files, and other design outputs are kept unique.*

If multiple Reconfigurable Partitions exist, then other configurations may be required. Additional configurations can also be created by importing previously implemented Reconfigurable Modules to create full designs that exist in hardware. This can be useful for creating full bitstreams with a desired combination for power-up, or for performing static timing analysis, power analysis, or simulation.

IMPORTANT: *See Known Issues in Chapter 8 for a current issue with reuse of implemented Reconfigurable Module checkpoints.*

Reporting

Each step of the implementation flow performs design rule checks (DRCs) unique to partial reconfiguration. Keep a close eye on the messages given by the implementation steps to ensure no critical warnings are issued. These messages provide guidance to optimize module interfaces, floorplans, and other key aspects of PR designs.

Most reports that can be generated do not have PR-specific sections, but useful information can be extracted nonetheless. For example, utilization information can be obtained by using the `-pblocks` switch for the `report_utilization` command. This shows the used and available resources within a given reconfigurable module. Here is an example using the design from the *Vivado Design Suite Tutorial: Partial Reconfiguration* (UG947) [Ref 1]:

```
report_utilization -pblocks [get_pblocks pblock_count]
```

For clock reporting, however, `report_clock_utilization` shows the clocks reserved for partial reconfiguration implementation.

Verifying Configurations

Once all configurations have been completely placed and routed, a final verification check can be done to validate consistency between these configurations using `pr_verify`. This command takes in multiple routed checkpoints (DCPs) as arguments, and outputs a log of any differences found in the static implementation and Partition Pin placement between them. Placement and routing within any RMs is ignored during the comparison.

When just two configurations are to be compared, list the two routed checkpoints as `<file1>` and `<file2>`. The `pr_verify` command loads both in memory and makes the comparison.

When more than two configurations are to be compared, provide a "master" configuration using the `-initial` switch, then list the remaining configurations by using the `-additional` switch, listing configurations in braces (`{` and `}`). The initial configuration is kept in memory and the remaining configurations are compared against the initial one. Bitstreams should not be generated for any configurations if any pair of configurations do not pass the PR Verify check.

```
pr_verify  [-full_check] [-file <arg>] [-initial <arg>] [-additional <arg>] [-quiet]
[-verbose] [<file1>] [<file2>]
```

Table 3-6: **`pr_verify` Options**

Command Option	Description
`-full_check`	Default behavior is to report the first difference only; if this option is selected, `pr_verify` reports all differences in placement or routing.
`-file`	Filename to output results to. Send output to console if `-file` is not used.
`-initial`	Select one routed design checkpoint against which all others will be compared.
`-additional`	Select one or more routed design checkpoints to compare against the initial one. List multiple checkpoints within braces, separated by a space, as in this example: `{config2.dcp config3.dcp config4.dcp}`
`-quiet`	Ignore command errors.
`-verbose`	Suspend message limits during command execution.

The following is a sample command line comparing two configurations:

```
pr_verify -full_check config1_routed.dcp config2_routed.dcp -file pr_verify_c1_c2.log
```

The following is a second example verifying three configurations:

```
pr_verify -full_check -initial config1.dcp -additional {config2.dcp config3.dcp} -file
three_config.log
```

The scripts provided with the *Vivado Design Suite Tutorial: Partial Reconfiguration* (UG947) [Ref 1] have a Tcl Proc called `verify_configs` that automatically runs all existing configurations through `pr_verify`, and reports if the DCPs are compatible or not.

Bitstream Generation

As in a flat flow, bitstreams are created with the `write_bitstream` command. For each design configuration, simply issue `write_bitstream` to create a full standard configuration file plus one partial bit file for each Reconfigurable Module within that configuration.

Xilinx recommends providing the configuration name and Reconfigurable Module names in the `-file` option specified with `write_bitstream`. Only the base bit file name can be modified, so it is important to record which Reconfigurable Modules were selected for each configuration.

Using the previous design, the following is an example of reading routed checkpoints (configurations) and creating bitstreams for all implemented Reconfigurable Modules.

```
open_checkpoint config1_routed.dcp
write_bitstream config1
```

This command creates a full design bitstream called `config1.bit`. This bitstream should be used to program the device from power-up and includes the functionality of any Reconfigurable Modules contained within. It also creates partial bit files `config1_pblock_rp1_partial.bit` and `config1_pblock_rp2_partial.bit` that can be used to reconfigure these modules while the FPGA continues to operate. Repeat these steps for each configuration.

TIP: *Rename each partial bit file to match the Reconfigurable Module instance from which it was built to uniquely identify these modules. The current solution names partial bit files only on the configuration base name and Pblock name: `<base_name>_<pblock_name>_partial.bit`*

If the full design configuration file is not required, then a single partial bitstream can be created on its own. With a full design configuration checkpoint loaded in memory, use the `-cell` option to identify the instance for which a partial bitstream is needed. The name of this partial bitstream can be given, as it is not automatically derived from the Pblock name.

```
write_bitstream -cell rp1 RM_count_down_partial.bit
```

This creates *only* a partial bitstream for the Reconfigurable Partition identified.

CAUTION! *Do not run `write_bitstream` directly on Reconfigurable Module checkpoints; only use full design checkpoints. Reconfigurable module checkpoints, while they are placed and routed submodules, have no information regarding the top level design implementation, and therefore would create unsuitable partial bit files.*

If a power-on configuration of the static design only is desired, run `write_bitstream` on the checkpoint that has empty Reconfigurable Partitions (after `update_design -black_box` and `update_design -buffer_ports` have run). This "black box configuration" can be compressed to reduce the bit file size and configuration time.

IMPORTANT: *The outputs of the RPs are undriven, so structure the design so that it powers up with the decoupling logic enabled.*

The size of each partial bitstream is reported in the output from `write_bitstream`. As this command is run, these messages will be reported for each partial and clearing bit file.

```
Creating bitmap...
Creating bitstream...
Partial bitstream contains 3441952 bits.
Writing bitstream ./Bitstreams/right_up_pblock_inst_shift_partial.bit...
```

Bitstream compression, encryption, and other advanced features can be used. See Known Limitations, page 108 for specific unsupported use cases for UltraScale devices.

Tcl Scripts

Scripts are provided to run this flow in the *Vivado Design Suite Tutorial: Partial Reconfiguration* (UG947) [Ref 1]. The details of these sample scripts are documented in the tutorial itself and in the readme.txt contained in the sample design archive.

Design Considerations and Guidelines for All Xilinx Devices

Overview

This chapter explains design requirements that are unique to Partial Reconfiguration, and covers specific PR features within the Xilinx® design software tools.

To take advantage of the Partial Reconfiguration capability of Xilinx FPGAs, you must analyze the design specification thoroughly, and consider the requirements, characteristics, and limitations associated with PR designs. This simplifies both the design and debug processes, and avoids potential future risks of malfunction in the design.

This chapter describes the design requirements that apply to all Xilinx 7 series and UltraScale™ devices. For design requirements specific to the individual FPGA and SoC architectures, see the following chapters in this manual:

- Chapter 5, Design Considerations and Guidelines for 7 Series and Zynq Devices
- Chapter 6, Design Considerations and Guidelines for UltraScale Devices

Design Hierarchy

Good hierarchical design practices resolve many complexities and difficulties when implementing a Partially Reconfigurable FPGA design. A clear design instance hierarchy simplifies physical and timing constraints. Registering signals at the boundary between static and reconfigurable logic eases timing closure. Grouping logic that is packed together in the same hierarchical level is necessary.

These are all well known design practices that are often not followed in general FPGA designs. Following these design rules is not strictly required in a partially reconfigurable design, but the potential negative effects of not following them are more pronounced. The benefits of Partial Reconfiguration are great, but the extra complexity in design could be more challenging to debug, especially in hardware.

For additional information about design hierarchy, see *Hierarchical Design Methodology Guide* (UG748) [Ref 11]

Dynamic Reconfiguration Using the DRP

Logic that is in the static region, and therefore is never partially reconfigured, can still be reconfigured dynamically through the Dynamic Reconfiguration Port (DRP). The DRP can be used to configure logic elements such as MMCMs, PLLs, and serial transceivers (MGTs).

Information about the DRP and dynamic reconfiguration, including how to use the DRP for specific design resources, can be found in these documents:

- *7 Series FPGAs Configuration User Guide* (UG470) [Ref 7]
- *7 Series FPGAs GTX/GTH Transceivers User Gui*de (UG476) [Ref 20]
- *7 Series FPGAs GTP Transceivers User Guide* (UG482) [Ref 21]
- *MMCM and PLL Dynamic Reconfiguration (7 Series)* (XAPP888) [Ref 22]
- *UltraScale Architecture Configuration User Guide* (UG570) [Ref 8]
- *UltraScale Architecture Clocking Resources User Guide* (UG572) [Ref 23]
- *UltraScale Architecture GTH Transceivers User Guide* (UG576) [Ref 24]
- *UltraScale Architecture GTY Transceivers User Guide* (UG578) [Ref 25]

Packing Logic

Any logic that must be packed together must be placed in the same group, whether it is static or reconfigurable. For example, if a LUT and a flip-flop are expected to be placed within the same slice, they must be within the same partition. Partition boundaries are barriers to optimization.

For Reconfigurable Partitions that include I/O, Clocking, and GT resources, it might be necessary to instantiate any I/O buffers that are automatically inferred by the tools inside that RP level. For example, if `GT_COMMON` is in an RP, the `IBUFDS_GTE` needs to be instantiated. If the associated I/O buffer is in the top-level/static portion, it cannot be packed.

Design Instance Hierarchy

The most simple method is to instantiate the Reconfigurable Partitions in the top-level module, but this is not required because a Reconfigurable Partition may be located in any level of hierarchy. Each Reconfigurable Partition must correspond to exactly one instance—an RP might not have more than one top. The instantiation has multiple modules with which it is associated.

Changes in design hierarchy can be used to merge and/or separate modules and leaf cells into and out of an RP level of hierarchy. There are several reasons to do this:

- To balance device resources between the PR region and Static region, making the design more efficient. For example, if the target RP takes up most of the device, and there is a module in the Static region that requires a high number of Block RAMs unavailable to Static, you can move that module into the PR region.
- If you need cells to reside in the same physical area of the device, but they are in a different design hierarchy. For example, if you need `GT_CHANNELS` to be placed in the same UltraScale Clock Region, but the design has GTs in both the Static and RP regions.
- To ensure that dedicated connections, for example from `IBUFDS_GT` to `GT_COMMON`, reside in the same region.

Reconfigurable Partition Interfaces

One of the fundamental requirements of a partially reconfigurable design is consistency between Reconfigurable Modules (RMs). As one module is swapped for another, the connections between the static design and the Reconfigurable Module must be identical, both logically and physically. To achieve this consistency, optimizations across the partition boundary or of the boundary itself are prohibited.

RECOMMENDED: *Keep interface logic connecting to and from RM ports consistent across all RMs. Do not change the levels of logic between RMs, such as using one level of logic in the initial design and five levels of logic in next RM. Also, do not change the driver type, such as using flip-flops in the initial RM and BRAM in next RM. Since the static side of the interface is locked after the initial configuration, the tools are unable to adjust for these changes in later configurations. Xilinx recommends registering all inputs and outputs of the RMs.*

For optimal efficiency, all ports of a Reconfigurable Partition should be actively used on the static design side. For example, if static drivers of the Reconfigurable Partition are driven by constants (0 or 1), they are implemented through the creation of a LUT instance and local tie-off to a constant driver and cannot be trimmed away. Likewise, unconnected outputs remain on Reconfigurable Partition outputs, creating unnecessary waste in the overall design. These measures must be taken by the implementation tools to ensure that all Reconfigurable Modules have the same port map during design assembly.

RECOMMENDED: *Examine the interface of all Reconfigurable Partitions after synthesis to ensure that as few constants or unconnected ports as possible remain. By clearing out dead logic, resource utilization is reduced, and congestion and timing closure challenges easier to address.*

Six different cases are possible for partition interface usage:

1. **Both Static and Reconfigurable Module sides have active logic.** (Applies to partition inputs or outputs)

   This is the optimal situation. A partition pin is inserted.

RECOMMENDED: *If partition inputs are driven by VCC or GND, push these constants into the Reconfigurable Modules. This reduces LUT usage and allow the implementation tools to optimize these constants with the RM logic.*

---

2. **The Static side has an active driver but the Reconfigurable Module does not have active loads.** (Applies to partition inputs)

   This is acceptable because it accommodates the situation in which not every Reconfigurable Module has the same I/O requirements. A partition pin is inserted, and the unused input ports are left unconnected.

   For example, one module might require CLK_A, while a second might require CLK_B. Clock spines are pre-routed to the Reconfigurable Partition clock regions, but the module only taps into the clock source that is needed. However, if a partition input is not used by any Reconfigurable Module, it should be removed from the partition instantiation.

3. **The Static side has active loads but the Reconfigurable Module does not have an active driver.** (Applies to partition outputs)

   This is acceptable and similar to the case above. A partition pin is inserted, and it is driven by ground (logic 0) within the Reconfigurable Module.

4. **The Static side does not have an active driver, but the Reconfigurable Module has active loads.** (Applies to partition inputs)

   This results in an error that must be resolved by modifying the partition interface.

   The following is an example of an error that may be seen for this scenario:

   ```
   ERROR: [Opt 31-65] LUT input is undriven either due to a missing connection from
   a design error, or a connection removed during opt_design.
   ```

   This error message would be followed by a LUT instance that is within the Reconfigurable Module.

5. **Reconfigurable Module has an active driver, but the Static side has no active loads.** (Applies to partition outputs)

   This does not result in an error, but is far from optimal because the RM logic remains. No partition pin is inserted. These partition outputs should be removed.

6. **Neither Static nor Reconfigurable Module sides have driver or loads for a partition port.** (Applies to partition inputs or outputs.)

   Nothing is inserted or used, so there is no implementation inefficiency, but it is unnecessary in terms of the instantiation port list.

Partition Pin Placement

Each pin of an RP has a partition pin (PartPin). By default the tools automatically place these PartPins inside of the RP Pblock range (which is required). For many cases, this automatic placement can be sufficient for the design. However, for timing-critical interface signals or designs with high congestion, it might be necessary to help guide the placement of the PartPins. The following is an example of how to achieve this:

- Define user `HD.PARTPIN_RANGE` constraints for some or all of the pins.

```
set_property HD.PARTPIN_RANGE {SLICE_Xx0Yx0:SLICE_Xx1Yy1
SLICE_XxNYyN:SLICE_XxMYyM} [get_pins <rp_cell_name>/*]
```

  By default the `HD.PARTPIN_RANGE` is set to the entire Pblock range. Defining a user range allows the tools to place PartPins in the specified areas, improving timing and/or reducing congestion.

IMPORTANT: *When examining the placement of PartPins, there are limited routing resources available along the edges, and especially in the corners, of the Pblock. The PartPin placer attempts to spread the partition pins, minimizing the number of partition pins per interconnect along the edges, and increasing the PartPin density towards the middle of the Pblock. When defining a custom `HD.PARTPIN_RANGE` constraint, be sure to make the range wide enough to allow for spreading, or you are likely to see congestion around the PartPins.*

Active-Low Resets and Clock Enables

In Xilinx 7 series FPGAs, there are no local inverters on control signals (resets or clock enables). The following description uses a reset as the example, but the same applies for clock enables.

If a design uses an active-Low reset, a LUT must be used to invert the signal. In non-PR designs that use all active-Low resets multiple LUTs are inferred but can be combined into a single LUT and pushed into the I/O elements (the LUT goes away). In non-PR designs that use a mix of High and Low, the LUT inverters can be combined into one LUT that remains in the design, but that has minimal effect on routing and the timing of the reset net (output of LUT can still be put on global resources). However, for a design that uses active-Low resets on a partition, it is possible to have inverters inferred inside the partition that cannot be

pulled out and combined. This makes it impossible to put the reset on global resources, and can lead to poor reset timing and to routing issues if the design is already congested.

The best way to avoid this is to avoid using active-Low control signals. However, there are cases where this is not possible (for example, when using an IP core with an Advanced eXtensible Interface (AXI) interface). In these cases the design should assign the active-Low reset to a signal at the top level, and use that new signal everywhere in the design.

As an example:

```
reset_n <= !reset;
```

Use the `reset_n` signal for all cases, and do not use the `!reset` assignments on signals or ports.

This ensures that a LUT is inferred only for the reset net for the whole design and has a minimal effect on design performance.

Decoupling Functionality

Because the reconfigurable logic is modified while the device is operating, the static logic connected to outputs of Reconfigurable Modules must ignore data from Reconfigurable Modules during Partial Reconfiguration. The Reconfigurable Modules do not output valid data until Partial Reconfiguration is complete and the reconfigured logic is reset. There is no way to predict or simulate the functionality of the reconfiguring module.

You must decide how the decoupling strategy is solved. A common design practice to mitigate this issue is to register all output signals (on the static side of the interface) from the Reconfigurable Module. An enable signal can be used to isolate the logic until it is completely reconfigured. Other approaches range from a simple 2-to-1 MUX on each output port, to higher level bus controller functions.

The static design should include the logic required for the data and interface management. It can implement mechanisms such as handshaking or disabling interfaces (which might be required for bus structures to avoid invalid transactions). It is also useful to consider the down-time performance effect of a PR module (that is, the unavailability of any shared resources included in a PR module during or after reconfiguration).

A Partial Reconfiguration Decoupler IP is available, allowing users to insert MUXes to efficiently decouple AXI Lite, AXI4-Stream, and custom interfaces. More information about the PR Decoupler IP is available on the Xilinx website.

Black Boxes

You can implement an RP as a *pseudo* black box. To do this, the RP must be a black box in the static design (either from bottom-up synthesis results or from running `update_design -black_box`). Then the black box can have LUT1 buffers placed on all inputs and outputs using the command `update_design -buffer_ports` on the black box RP cell:

```
update_design -cell <rp_cellName> -buffer_ports
```

Now you can run this design through implementation to place and route the LUT1 buffers (and static logic, if not already placed and routed).

All the inserted LUT1 output buffers are tied to a logic 0 (ground). If it is necessary to drive a logic 1 (V_{CC}) from the RP outputs, this can be controlled using an RP pin property called `HD.PARTPIN_TIEOFF`. This property can be set at any time (all the way up to pre-write_bitstream), and it controls the LUT equation of the LUT1 buffer connected to the specified port. The default value is '`0`', which configures the LUT as a route-thru (output is 0). Setting this property to '`1`' configures the LUT as an inverter (output is 1). You might have to change the output value in some design situations.

```
set_property HD.PARTPIN_TIEOFF 1 [get_pins <RP_cellName>/<output_pinName>]
```

The pseudo black box has no user logic (just the tool-inserted LUT1 buffers). The black box bitstream contains information for these LUTs, as well as any static logic/routes that use resources inside the RP frames. Static routes that pass through the region, including interface nets up to the partition pin nodes, exist within this region. Programming information for these signals is included in the black box programming bitstream.

Use of black boxes is an effective way to reduce the size of a full configuration BIT file, and therefore reduce the initial configuration time. The compression feature might also be enabled to reduce the size of BIT files. This option looks for repeated configuration frame structures to reduce the amount of configuration data that must be stored in the BIT file. The compression results in reduced configuration and reconfiguration time. When the compression option is applied to a routed PR design, all of the BIT files (full and partial) are created as compressed BIT files. To enable compression, set this property prior to running `write_bitstream`:

```
set_property BITSTREAM.GENERAL.COMPRESS TRUE [current_design]
```

Effective Approaches for Implementation

There are trade-offs associated with optimizing any FPGA design. Partial Reconfiguration is no different. Partitions are barriers to optimization, and reconfigurable frames require specific layout constraints. These are the additional costs to building a reconfigurable design. The additional overhead for timing and area needs vary from design to design. To minimize the impact, follow the design considerations stated in this guide.

When building Configurations of a reconfigurable design, the first Configuration to be chosen for implementation should be the most challenging one. Be sure that the physical region selected has adequate resources (especially elements such as block RAM and DSP48) for each Reconfigurable Module in each Reconfigurable Partition, then select the most demanding (in terms of either timing or area) RM for each RP. If all of the RMs in the subsequent Configurations are smaller or slower, it is easier to meet their demands. Timing budgets should be established to meet the needs of all Reconfigurable Modules.

If it is not clear which reconfigurable module is the most challenging, each can be implemented in parallel in context with static, allowing static to be placed and routed for each. Examine resource utilization statistics and timing reports to see which configuration met design criteria most easily and which had the tightest tolerances, or which missed by the widest margins.

IMPORTANT: *Focus attention on the configuration that is the furthest from meeting its goals, iterating on design sources, constraints, and strategies until needs are met. At some point, one configuration must be established as the golden result for the static design, and that implementation of the static logic will be used for all other configurations.*

Building Up Implementation Requirements

Implementation of Partial Reconfiguration designs requires that certain fundamental rules are followed. These rules have been established to ensure that a partial bitstream can be accurately created and safely delivered to an active device. As noted throughout this document, these rules include these basic premises:

- The logical and physical interface of a Reconfigurable Partition remains consistent as each Reconfigurable Module is implemented.
- The logic and routing of a Reconfigurable Module is fully contained within a physical region which is then translated into a partial bitstream.
- The logic of the static design must be kept out of the reconfigurable region if the dedicated initialization feature is used.

These requirements necessitate specific implementation rules for optimization, placement and routing. Application of these rules might make it more difficult to meet design goals, including timing closure. A recommended strategy is to build up this set of requirements one at a time, allowing you to analyze the results at each step. Starting with the most challenging configuration and the full set of timing constraints, implement the design through place and route and examine the results, making sure you have sufficient timing slack and resources available to continue to the next step.

1. Implement the design with no Pblocks. Use bottom-up synthesis and follow general Hierarchical Design recommendations, such as registered boundaries, to achieve a baseline result.
2. Add Pblocks for the design partitions that will later be marked reconfigurable. This floorplan can be based on the results established in the bottom-up synthesis run from Step 1. Logic from the Reconfigurable Modules must be placed in the Pblocks, but static logic may be included there as well.

   While creating these Pblocks, the `HD.RECONFIGURABLE` property (and optionally, the `RESET_AFTER_RECONFIG` property) can be added temporarily to run PR-specific Design Rule Checks. This ensures that the floorplan created meets PR size and alignment requirements.
3. With the floorplan established, separate the placement of static design resources from those to be reconfigurable by adding the `EXCLUSIVE_PLACEMENT` property to the Pblocks. This keeps static logic placed outside the defined Pblocks.
4. Keep the routing for Reconfigurable Modules bound within the Pblocks by applying the `CONTAIN_ROUTING` property to the Pblocks. With the properties from this and the previous step, the only remaining rules relate to boundary optimization procedures as well as PR-specific Design Rule Checks.
5. Finally, mark the Reconfigurable Partition Pblocks as `HD.RECONFIGURABLE`. The `EXCLUSIVE_PLACEMENT` and `CONTAIN_ROUTING` properties are now redundant and can be removed.

If design requirements are not met at any of these steps, you have to opportunity to review the design structure and constraints in light of the newly applied implementation condition.

Defining Reconfigurable Partition Boundaries

Partial reconfiguration is done on a frame-by-frame basis. As such, when partial BIT files are created, they are built with a discrete number of configuration frames. The size of a partial bit file depends on the number and type of frames included. You can see this size in the header of a raw bit file (`.rbt`) created by `write_bitstream -rawbitfile`.

Partition boundaries do not have to align to reconfigurable frame boundaries, but the most efficient place and route results are achieved when this is done. Static logic is permitted to exist in a frame that will be reconfigured, as long as:

- It is outside the area group defined by the Pblock, and
- It does not contain dynamic elements such as Block RAM, Distributed (LUT) RAM, or SRLs. (7 series only)

When static logic is placed in a reconfigured frame, the exact functionality of the static logic is rewritten, and is guaranteed not to glitch.

Irregular shaped Partitions (such as a T or L shapes) are permitted but discouraged. Placement and routing in such regions can become challenging, because routing resources must be entirely contained within these regions. Boundaries of Partitions can touch, but this is not recommended, as some separation helps mitigate potential routing restriction issues as these partitions connect to the static design. Nested or overlapping Reconfigurable Partitions (Partitions within Partitions) are not permitted. Design rule checks (**Tools > Report > Report DRC**) validate the Partitions and settings in a PR design.

Only one Reconfigurable Partition can exist per physical Reconfigurable Frame.

A Reconfigurable Frame is the smallest size physical region that can be reconfigured, and its height aligns with clock region or I/O bank boundaries. A Reconfigurable Frame cannot contain logic from more than one Reconfigurable Partition. If it were to contain logic from more than one Reconfigurable Partition, it would be very easy to reconfigure the region with information from an incorrect Reconfigurable Module, thus creating contention. The software tools are designed to avoid that potentially dangerous occurrence.

Avoiding Deadlock

Some transactions across an RM boundary can take multiple cycles to complete. Removing an RM after a transaction has started but before it completes causes the system to deadlock (for example, the master, which initiated the transaction, waits for a response from a slave which no longer exists).

Additionally, the RM itself can cause deadlock. For example, assume some software is polling an RM register for a particular value. If the RM is removed, the software might stall as it continues to wait. It could also stall while waiting on a large block transfer to complete.

Any Partial Reconfiguration design should be built with some sort of handshaking, ensuring that the removal of a Reconfigurable Module occurs when it is safe to do so. This request or acknowledgement pairing is part of the user design and can be built in any fashion you deem appropriate.

Design Revision Checks

A partial bitstream contains programming information and little else, as described in Chapter 7, Configuring the Device. While you do not need to identify the target location of the bitstream (the die location is determined by the addressing that is part of the BIT file), there are no checks in the hardware to ensure the partial bitstream is compatible with the currently operating design. Loading a partial bitstream into a static design that was not implemented with that Reconfigurable Module revision can lead to unpredictable behavior.

Xilinx suggests that you prefix a partial bitstream with a unique identifier indicating the particular design, revision and module that follows. This identifier can be interpreted by your configuration controller to verify that the partial bitstream is compatible with the resident design. A mismatch can be detected, and the incompatible bitstream can be rejected, before being loaded into configuration memory. This functionality must be part of your design, and would be similar to or in conjunction with decryption and/or CRC checks, as described in *PRC/EPRC: Data Integrity and Security Controller for Partial Reconfiguration* (XAPP887) [Ref 26].

A bitstream feature provides a simple mechanism for tagging a design revision. The `BITSTREAM.CONFIG.USR_ACCESS` property allows you to enter a revision ID directly into the bitstream. This ID is placed in the `USR_ACCESS` register, accessible from the FPGA programmable logic through a library primitive of the same name. Partial Reconfiguration designs can read this value and compare it to information a user can add to a header of a partial bitstream to confirm the revisions of the design match. More information on this switch can be found at this link in the *Vivado Design Suite User Guide: Programming and Debugging* (UG908) [Ref 27].

CAUTION! *Do not use the TIMESTAMP feature because this value is not consistent for each call to `write_bitstream`. Only select a consistent, explicit ID to be used for all `write_bitstream` runs.*

Simulation and Verification

Configurations of Partial Reconfiguration designs are complete designs in and of themselves. All standard simulation, timing analysis, and verification techniques are supported for PR designs. Partial reconfiguration itself cannot be simulated. Specifically, the delivery of a partial bitstream to a configuration port like the ICAP to see the resulting change (including intermediate states) in a Reconfigurable Partition.

Design Considerations and Guidelines for 7 Series and Zynq Devices

Overview

This chapter explains design requirements that are unique to Partial Reconfiguration, and are specific to 7 series and Zynq®-7000 AP SoC devices.

To take advantage of the Partial Reconfiguration capability of Xilinx devices, you must analyze the design specification thoroughly, and consider the requirements, characteristics, and limitations associated with PR designs. This simplifies both the design and debug processes, and avoids potential future risks of malfunction in the design.

Design Elements Inside Reconfigurable Modules

Not all logic is permitted to be actively reconfigured. Global logic and clocking resources must be placed in the static region to not only remain operational during reconfiguration, but to benefit from the initialization sequence that occurs at the end of a full device configuration.

Logic that can be placed in a Reconfigurable Module includes:

- All logic components that are mapped to a CLB slice in the device. This includes LUTs (look-up tables), FFs (flip-flops), SRLs (shift registers), RAMs, and ROMs.
- Block RAM and FIFO:
  - RAMB18E1, RAMB36E1, BRAM_SDP_MACRO, BRAM_SINGLE_MACRO, BRAM_TDP_MACRO
  - FIFO18E1, FIFO36E1, FIFO_DUALCLOCK_MACRO, FIFO_SYNC_MACRO

  Note: The IN_FIFO and OUT_FIFO design elements cannot be placed in an RM. These design elements must remain in static logic.
- DSP blocks: DSP48E1
- PCIe® (PCI Express): Entered using PCIe IP

All other logic must remain in static logic and must not be placed in an RM, including:

- Clocks and Clock Modifying Logic - Includes BUFG, BUFR, MMCM, PLL, and similar components
- I/O and I/O related components (ISERDES, OSERDES, IDELAYCTRL, etc.)
- Serial transceivers (MGTs) and related components
- Individual architecture feature components (such as BSCAN, STARTUP, XADC, etc.)

Global Clocking Rules

Because the clocking information for every Reconfigurable Module for a particular Reconfigurable Partition is not known at the time of the first implementation, the PR tools pre-route each BUFG output driving a partition pin on that RP to all clock regions that the Pblock encompasses. This means that clock spines in those clock regions might not be available for static logic to use, regardless of whether the RP has loads in that region.

In 7 series devices, up to 12 clock spines can be pre-routed into each clock region. This limit must account for both static and reconfigurable logic. For example, if 3 global clocks route to a clock region for static needs, any RP that covers that clock region can use the 9 global clocks available, collectively, in addition to those three top-level clocks.

In the example shown in Figure 5-1, page 60, `icap_clk` is routed to clock regions X0Y1, X0Y2, and X0Y3 prior to placement, and static logic is able to use the other clock spines in that region.

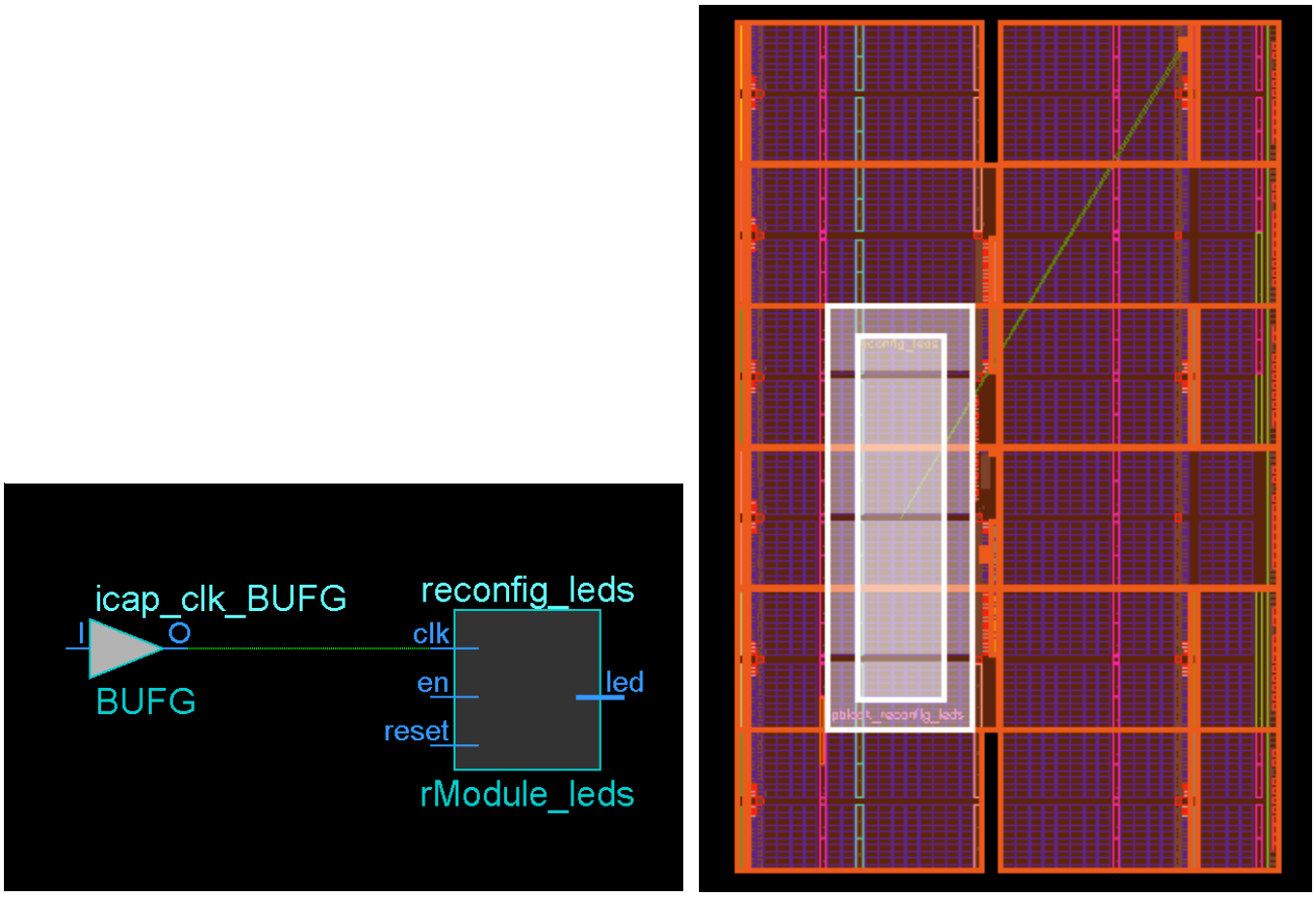

Figure 5-1: **Pre-routing Global Clock to Reconfigurable Partition**

RECOMMENDED: *If there are a large number of global clocks driving an RP, create area groups that encompass complete clock regions to ease placement and routing of static logic. Global clocks can be downgraded to regional clocks (for example, BUFR, BUFH) for clocks with fewer loads or less demanding requirements. Shifting clocks from global to local resources allows for more flexibility in floorplanning when the RP requires many unique clocks.*

Creating Pblocks for 7 Series Devices

As noted in Apply Reset After Reconfiguration in Chapter 3, the height of the Reconfigurable Partition must align to clock region boundaries if `RESET_AFTER_RECONFIG` is to be used. Otherwise, any height can be selected for the Reconfigurable Partition.

The width of the Reconfigurable Partition must be set appropriately to make most efficient usage of interconnect and clocking resources. The left and right edges of Pblock rectangles should be placed between two resource columns (for example, CLB-CLB, CLB-BRAM or CLB-DSP) and not between two interconnect columns (INT-INT). This allows the placer and router tools the full use of all resources for both static and reconfigurable logic. Implementation tool DRCs provide guidance if this approach is not followed.

Automatic Adjustments for Reconfigurable Partition Pblocks

The Pblock `SNAPPING_MODE` property automatically resizes Pblocks to ensure no back-to-back violations occur for 7 series designs. When `SNAPPING_MODE` is set to a value of `ON` or `ROUTING`, it creates a new set of derived Pblock ranges that are used for implementation. The new ranges are stored in memory, and are not written out to the XDC. Only the `SNAPPING_MODE` property is written out, in addition to the normal Pblock constraints.

In 7 series devices the structure is such that the routing resources, called interconnect tiles, are placed adjacent, or back-to-back. When floorplanning for partial reconfiguration, it is important to understand where these back-to-back boundaries exist. If a Pblock splits these paired interconnect tiles, it is called a back-to-back violation. For more information on back-to-back interconnect please refer to Creating Reconfigurable Partition Pblocks Manually, page 64.

The original Pblock rectangle(s) are not modified when using `SNAPPING_MODE` and can still be resized, moved, or extended with additional rectangles. Whenever the original Pblock rectangle is modified, the derived ranges are automatically recalculated. The `SNAPPING_MODE` property is supported in batch mode, so there is no requirement to open the current Pblock in the Vivado® IDE to set the `SNAPPING_MODE` value, although this option is available when performing interactive floorplanning, as shown in Figure 5-2, page 62.

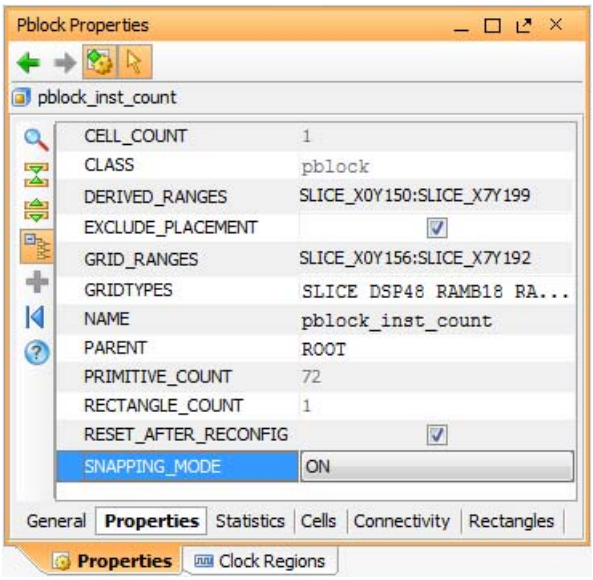

Figure 5-2: **Enabling the SNAPPING_MODE Property in the Vivado IDE**

When you set the SNAPPING_MODE property using the following syntax (or by selecting the Pblock Property as shown above), the implementation tools automatically see the corrected Pblock ranges.

```
set_property SNAPPING_MODE ON [get_pblocks <pblock_name>]
```

The table below shows SNAPPING_MODE property values for 7 series devices.

Table 5-1: **SNAPPING_MODE Property Values for 7 Series Devices**

Property	Value	Description
SNAPPING_MODE	OFF	Default for 7 series. No adjustments are made and DERIVED_RANGES == GRID_RANGES
	ON	Fixes all back-to-back violations.
	ROUTING	Same behavior as ON, except for the following exceptions: • Does not fix back-to-back violations across the center clock column to improve routing. • Grabs unbonded I/O and GT sites that are within or adjacent to the RP Pblock to improve routing. It can only use these resources for PR routing if they sites are unbonded and if the entire column (Clock Region in height) are included in the Pblock rectangle. This is the recommended value for 7 series and Zynq designs.

The SNAPPING_MODE property also works in conjunction with RESET_AFTER_RECONFIG. Using RESET_AFTER_RECONFIG requires Pblocks to be vertically frame (or clock region) aligned. When SNAPPING_MODE is set to ON or to ROUTING and RESET_AFTER_RECONFIG is set to TRUE, the derived ranges automatically include all sites necessary to meet this requirement.

Figure 5-3 shows the original user-created pblock in purple. RESET_AFTER_RECONFIG has been enabled, and both left and right edges split interconnect columns. By applying SNAPPING_MODE, the resulting derived Pblock (shown in yellow) is *narrower* to avoid INT-INT boundaries, and *taller* to snap to the height of a clock region.

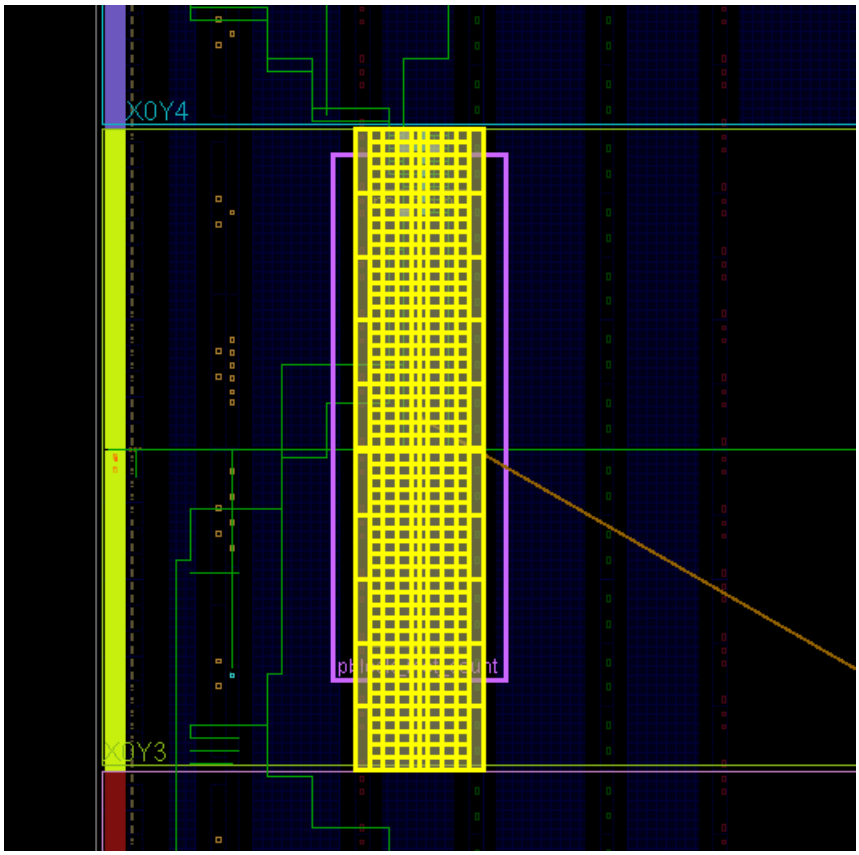

Figure 5-3: **Original and Derived Pblocks using SNAPPING_MODE**

Creating Reconfigurable Partition Pblocks Manually

If automatic modification to the Reconfigurable Partition Pblock is not desired to fix back-to-back issues, you can create Pblock ranges manually to meet your needs. This is most useful when explicit control is needed for Pblocks that must span non-reconfigurable sites, such as configuration blocks or the center column, which contains clock buffer resources.

In Figure 5-4, note that the left and right edges are drawn between CLB columns for the Pblock highlighted in white. Visualization of the interconnect tiles as shown in this image requires that the routing resources are turned on, using this symbol in the Device View: .

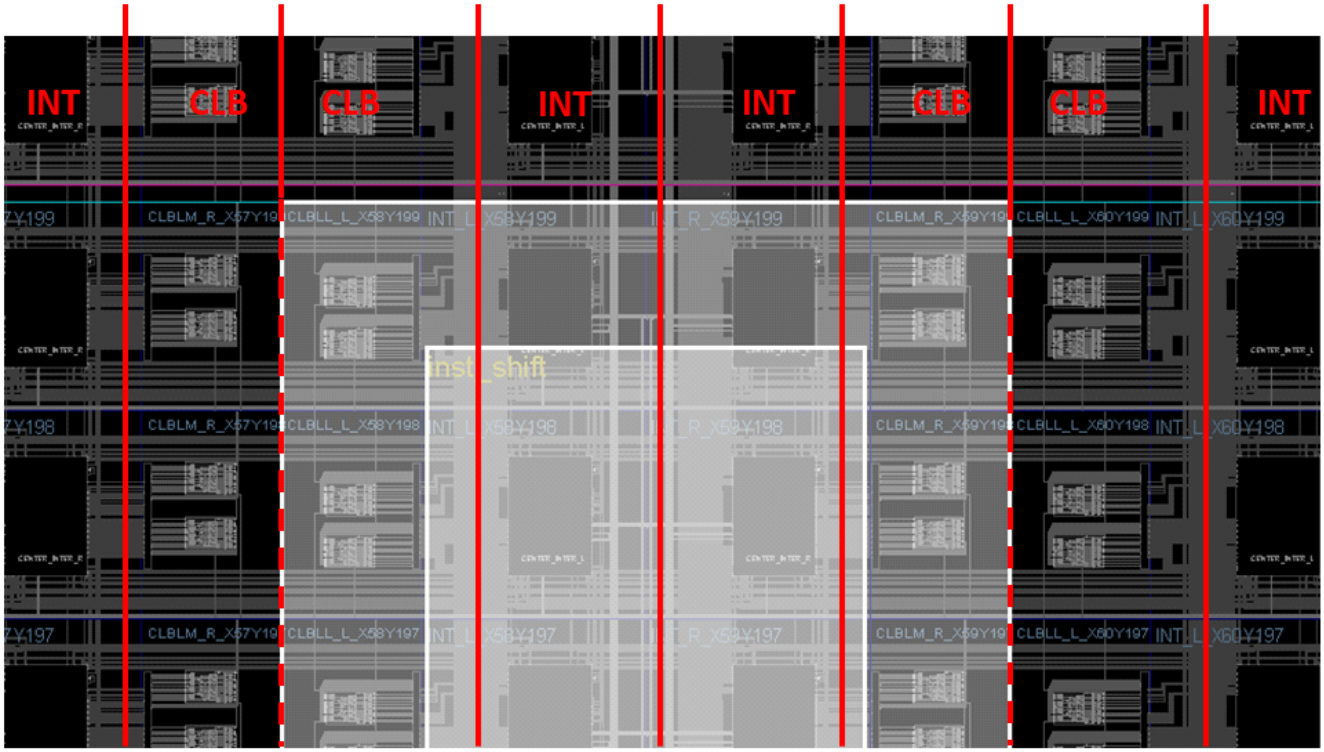

Figure 5-4: **Optimal - Reconfigurable Partition Pblock Splitting CLB-CLB on Both Left and Right Edges**

The Reconfigurable Partition Pblock must include all reconfigurable element types within the shape drawn. In other words, if the rectangle selected encompasses CLB (Slice), block RAM, and DSP elements, all three types must be included in the Pblock constraints. If one of these is omitted, a DRC is triggered with an alert that a split interconnect situation has been detected.

Other considerations must be taken if the Reconfigurable Partition spans non-reconfigurable sites, such as the center-column clocking resources or configuration components (ICAP, BSCAN, etc.), or abuts non-reconfigurable components such as I/O. If a Pblock edge splits interconnect columns for different resource types, implementation tools accept this layout, but restrict placement in the columns on each side of the boundary. If this prohibits sites that are needed for the design (such as the ICAP or BSCAN, for example), the Pblock must be broken into multiple rectangles to clearly define reconfigurable logic usage, or `SNAPPING_MODE` must be used.

The implementation tools automatically prevent placement on both sides of the back-to-back interconnect by creating `PROHIBIT` constraints. If the sites that are prohibited due to a back-to-back violation are not needed in the design, it is acceptable to leave the back-to-back violation in the design. Doing so allows an extra column of routing tiles to be included in the PR region, and can reduce congestion in a PR region that spans non-reconfigurable sites. In this case, a Critical Warning is issued by DRCs, but the warning can be safely ignored if you understand the trade-offs of placement versus routing resources.

The one exception to this behavior is around the clock column. If a violation occurs at the clock column boundary, `PROHIBIT` constraints are generated for the RM side of the violation (typically SLICE prohibits), but the clocking resources do not get prohibit constraints and are still available to the static logic. The reason `SNAPPING_MODE` has a value of `ROUTING` is to take advantage of this special exception. The `SNAPPING_MODE` property has a value of `ROUTING` is to take advantage of this special exception. For example, the initial floorplan shown in Figure 5-5, page 66 spans the center column, which contains clock buffer resources (BUFHCE/BUFGCTRL). These resources have not been included in the Pblock, as they are not highlighted in Figure 5-5. There is violation caused by spanning this clock column but the resources can still be used by the static logic.

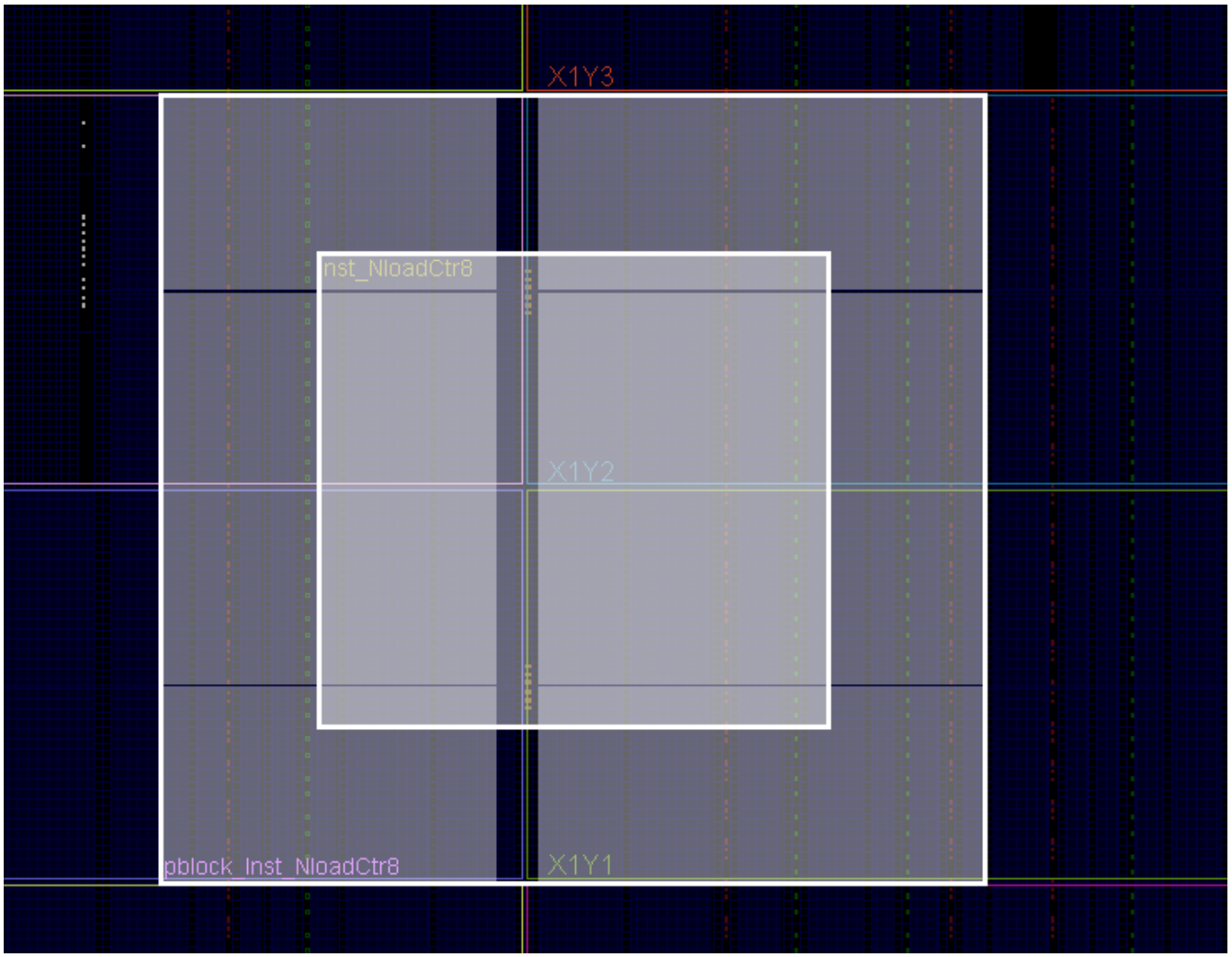

Figure 5-5: **Pblock Spanning Non-Reconfigurable Sites**

Prohibited sites appear in placed or routed checkpoints as sites with a red circle with a slash, as shown in Figure 5-6, page 67. With this automatic prohibit feature, the routing interconnect associated with reconfigurable sites (CLBs) can still be used for the reconfigurable module even though the CLBs themselves are not used. In Figure 5-6, the column of INT on the left are available for the RM, but the column of INT on the right is only available for static logic because these are part of the clock tile, which is not reconfigurable for 7 series devices.

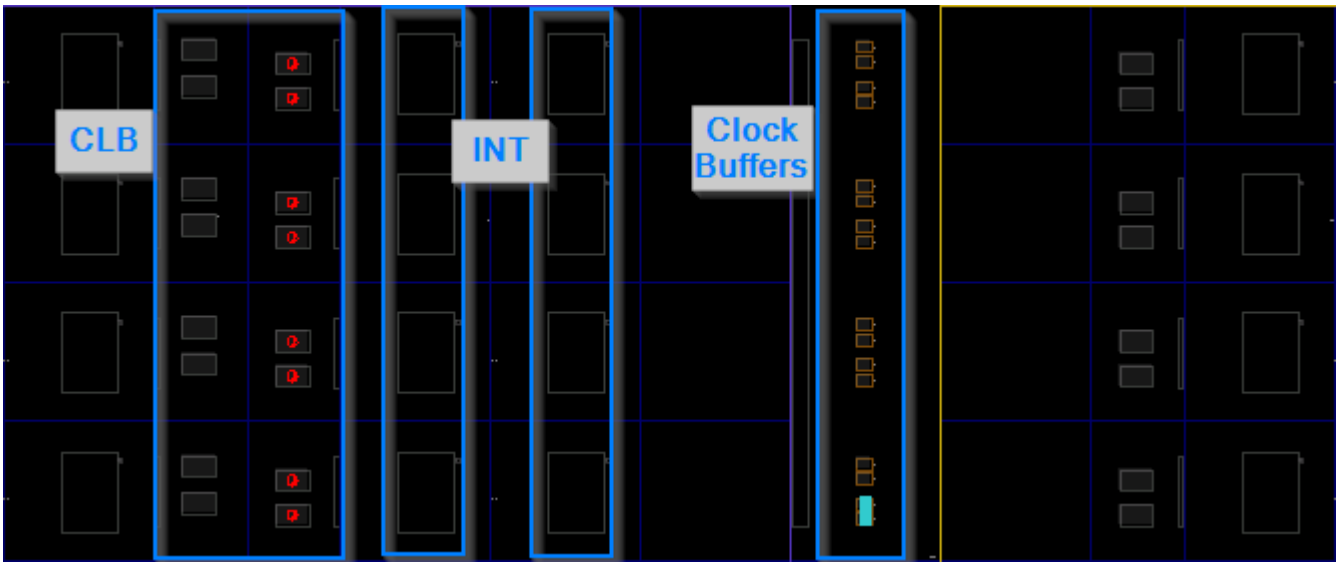

Figure 5-6: **Prohibited Sites in a Checkpoint**

If a back-to-back violation prohibits sites that are needed for the design (that is, ICAP or BSCAN sites), a placement error is issued, stating that not enough sites are available in the device.

```
ERROR: [Common 17-69] Command failed: Placer could not place
all instances
```

To avoid this restriction, create multiple Pblock rectangles that avoid splitting interconnect columns, as shown in Figure 5-7, page 68, or use the Pblock `SNAPPING_MODE` property.

RECOMMENDED: *In general, spanning non-reconfigurable site types (such as IOB, configuration, or clocking columns) should be avoided whenever possible. If the Pblock must span one of these, the clocking column is the least risky choice, owing to its special nature (described previously). Use `SNAPPING_MODE ROUTING` to cross this boundary as efficiently as possible.*

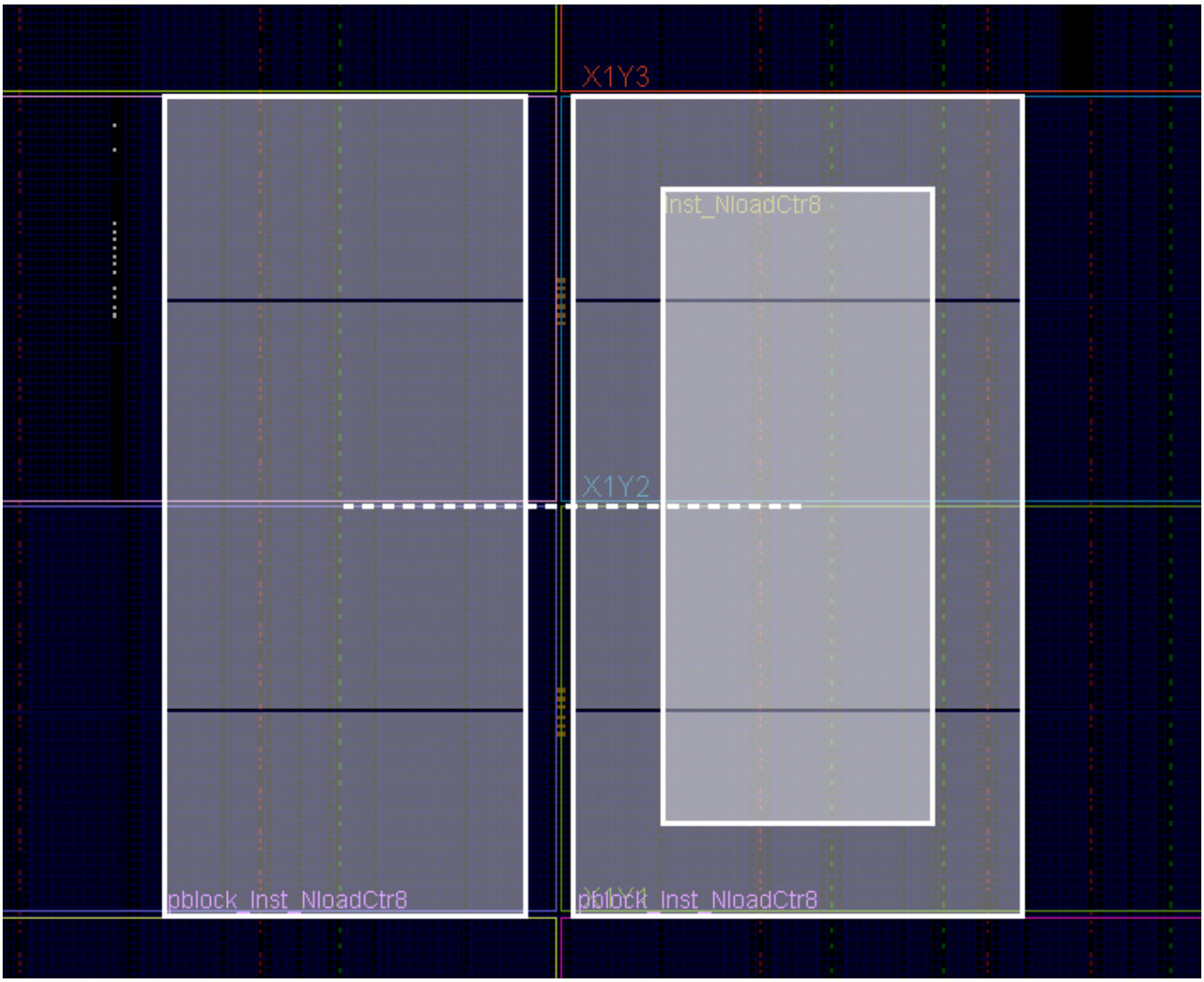

Figure 5-7: **Multiple Pblock Rectangles that Avoid Non-Reconfigurable Resources**

Figure 5-8 is a close-up of this split, showing Slice (CLB) and Interconnect (INT) resource types. The gap between the two Pblock rectangles gives full access to the BUFHCE components to route completely using static resources. This also leaves one column of CLBs available for the static design to use. Although routing resources exist that can cross these gaps, the overall routability of such structures is notably reduced. This approach is more challenging and should be avoided if possible. When spanning other static boundaries, such as IOB or configuration tiles, the routing gap for the PR region becomes two INT resources, and routing becomes difficult.

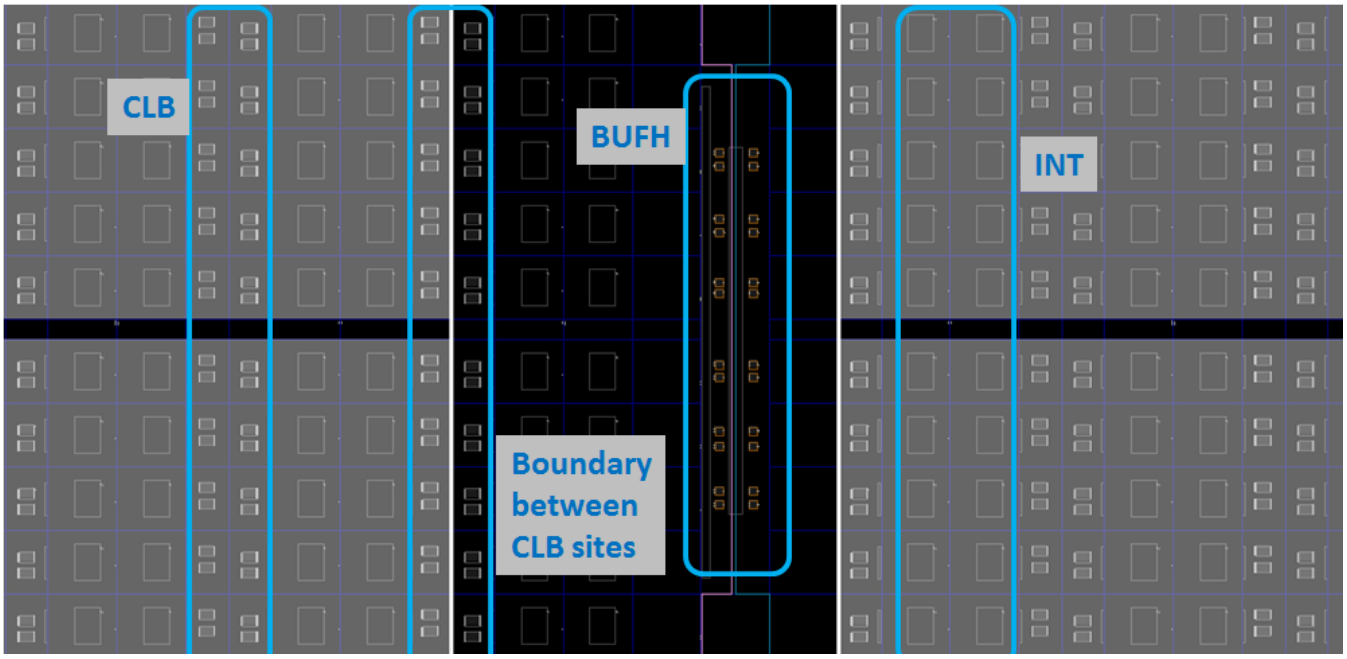

Figure 5-8: **Close-up Showing Columns Reserved for Clock Routing Usage**

Irregular shaped Partitions (such as a T or L shapes) are permitted, but you are encouraged to keep overall shapes a simple as possible. Placement and routing in such regions can become challenging because routing resources must be entirely contained within these regions. Boundaries of Partitions can touch, but this is not recommended, as some separation helps mitigate potential routing restriction issues. Nested or overlapping Reconfigurable Partitions (Partitions within Partitions) are not permitted.

Finally, only one Reconfigurable Partition can exist per physical Reconfigurable Frame. A Reconfigurable Frame is the smallest size physical region that can be reconfigured, and aligns with clock region boundaries. A Reconfigurable Frame cannot contain logic from more than one Reconfigurable Partition. If it were to contain logic from more than one Reconfigurable Partition, it would be very easy to reconfigure the region with information from an incorrect Reconfigurable Module, thus creating contention. The Vivado tools are designed to avoid that potentially dangerous occurrence.

Using High Speed Transceivers

Xilinx high speed transceivers (GTP, GTX, GTH,GTZ) are not reconfigurable in 7 series devices, and must remain in static logic. However, settings for the transceivers can be updated during operation using the DRP ports. For more information on the transceiver settings and DRP access, see *7 Series FPGAs GTX/GTH Transceivers User Guide* (UG476) [Ref 20], or *7 Series FPGAs GTP Transceivers User Guide* (UG482) [Ref 21].

Partial Reconfiguration Design Checklist (7 Series)

Xilinx highly encourages the following items for a 7 series FPGA design using Partial Reconfiguration:

Recommended Clocking Networks

Are you using Global Clock Buffers, Regional Clock Buffers, or Clock Modifying Blocks (MMCM, PLL)?

These blocks must be in static logic.

See Design Elements Inside Reconfigurable Modules, page 58 for more information, and Global Clocking Rules, page 59 for complete details on global clock implementation.

Configuration Feature Blocks

Are you using device feature blocks (BSCAN, CAPTURE, DCIRESET, FRAME_ECC, ICAP, STARTUP, USR_ACCESS)?

These featured blocks must be in static logic.

See Design Elements Inside Reconfigurable Modules, page 58 for more information.

High Speed Transceiver Blocks

Do you have high speed transceivers in your design?

High speed transceivers must remain in the static partition.

See Using High Speed Transceivers, page 70 for specific requirements.

System Generator DSP Cores, HLS cores, or IP Integrator Block Diagrams

Are you using System Generator DSP cores, HLS cores, or IP Integrator block diagrams in your Partial Reconfiguration design?

Any type of source can be used as long as it follows the fundamental requirements for Partial Reconfiguration. Any code processed by SysGen, HLS, or Vivado IP Integrator (or other tools) is eventually synthesized. The resulting design checkpoint or netlist must be made up entirely of reconfigurable elements (CLB, block RAM, DSP) for it to be legally included in an RP.

Packing I/Os into Reconfigurable Partitions

Do you have I/Os in reconfigurable modules?

All I/Os must reside in static logic.

See Design Elements Inside Reconfigurable Modules, page 58 for more information.

Packing Logic into Reconfigurable Partitions

Is all logic that must be packed together in the same Reconfigurable Partition?

Any logic that must be packed together must be in the same RP and RM.

See Packing Logic, page 48 for more information.

Packing Critical Paths into Reconfigurable Partitions

Are critical paths contained within the same partition?

Reconfigurable partition boundaries limit some optimization and packing, so critical paths should be contained within the same partition.

See Packing Logic, page 48 for more information.

Floorplanning

Can your Reconfigurable Partitions be floorplanned efficiently?

See Creating Pblocks for 7 Series Devices, page 61 for more information.

Recommended Decoupling Logic

Have you created decoupling logic on the outputs of your RMs?

During reconfiguration the outputs of RPs are in an indeterminate state, so decoupling logic must be used to prevent static data corruption.

See Decoupling Functionality, page 52 for more information.

Recommended Reset after Reconfiguration

Are you resetting the logic in an RM after reconfiguration?

After reconfiguration, new logic might not start at its initial value. If the Reset After Reconfiguration property is not used, a local reset must be used to ensure it comes up as expected when decoupling is released. Clock and other inputs to the reconfigurable partition can also be disabled during reconfiguration to prevent initialization issues. Alternatively, the Reset After Reconfiguration property can be applied. This option holds internal signals steady during reconfiguration, then issues a masked global reset to the reconfigured logic.

See Apply Reset After Reconfiguration in Chapter 3 for more information.

Debugging with Logic Analyzer Blocks

Are you using the Vivado Logic Analyzer with your Partial Reconfiguration design?

Vivado Logic Analyzer (ILA/VIO debug cores) can be used in your Partial Reconfiguration design, but they must be in static logic.

Efficient Reconfigurable Partition Pblocks

Have you created efficient Reconfigurable Partition Pblock(s) for your design?

The height of the Reconfigurable Partition Pblock must align with the top and bottom of a clock region boundary, if the `RESET_AFTER_RECONFIG` property is to be used. Otherwise, any height can be selected for the Reconfigurable Partition Pblock.

See Creating Pblocks for 7 Series Devices, page 61 for more information.

Validating Configurations

How do you validate consistency between configurations?

The `pr_verify` command is used to make sure all configurations have matching imported resources.

See Verifying Configurations in Chapter 3 for more information.

Configuration Requirements

Are you aware of the particular configuration requirements for Partial Reconfiguration for your design and device?

Each device family has specific configuration requirements and considerations.

See the Chapter 7, Configuring the Device for more information.

Effective Pblock recommendations

Does an RP Pblock extend over the center clock column or the configuration column in the device?

Due to the back-to-back INT tile requirement for 7 series devices, coupled with the `CONTAIN_ROUTING` requirement, extending a Pblock over these specialized blocks in the device can make routing very difficult or impossible. Avoid extending an RP Pblock across these areas whenever possible.

See Automatic Adjustments for Reconfigurable Partition Pblocks, page 61 and Creating Reconfigurable Partition Pblocks Manually, page 64 for more information on back-to-back requirements.

XILINX
ALL PROGRAMMABLE™

Chapter 6

Design Considerations and Guidelines for UltraScale Devices

Overview

This chapter explains design requirements that are unique to Partial Reconfiguration, and are specific to UltraScale™ devices.

To take advantage of the Partial Reconfiguration capability of Xilinx devices, you must analyze the design specification thoroughly, and consider the requirements, characteristics, and limitations associated with PR designs. This simplifies both the design and debug processes, and avoids potential future risks of malfunction in the design.

Design Elements Inside Reconfigurable Modules

In UltraScale devices, nearly all component types may be partially reconfigured.

Logic that can be placed in a Reconfigurable Module includes:

- All logic components that are mapped to a CLB slice in the FPGA. This includes LUTs (look-up tables), FFs (flip-flops), SRLs (shift registers), RAMs, and ROMs.
- Block RAM (BRAM) and FIFO: RAMB18E2, RAMB36E2, FIFO18E2, FIFO36E2
- DSP blocks: DSP48E2
- PCIe® (PCI Express), CMAC (100G MAC), and ILKN (Interlaken MAC) blocks
- SYSMON (XADC and System Monitor)
- Clocks and Clock Modifying Logic: Includes BUFG, BUFGCE, BUFGMUX, MMCM, PLL, and similar components
- I/O and I/O related components (ISERDES, OSERDES, IDELAYCTRL, etc.)
- Serial transceivers (MGTs) and related components

Only configuration components must remain in the static part of the design. These components are:

- BSCAN
- CFG_IO_ACCESS
- DCIRESET
- DNA_PORT
- EFUSE_USR
- FRAME_ECC
- ICAP
- MASTER_JTAG
- STARTUP
- USR_ACCESS

Creating Pblocks for UltraScale Devices

As part of improvements to the UltraScale architecture, the smallest unit that can be reconfigured is much smaller than in previous architectures. The minimum required resources for reconfiguration varies based on the resource type, and are referred to as a Programmable Unit (PU). Because adjacent sites share a routing resource (or Interconnect Tile) in UltraScale, a PU is defined in terms of pairs.

Examples of some of the minimum PU that can be reconfigured based on the site types:

- CLB PU: 2 adjacent CLBs, and the shared interconnect
- Block RAM PU: 1 BRAM/FIFO, the 5 adjacent CLBs, and the shared interconnect
- DSP PU: 1 DSP, the 5 adjacent CLBs, and the shared interconnect
- IOB PU: The IO of the full height of the clock_region and includes BITSLICE_CONTROL, BITSLICE_RX_TX, BITSLICE_TX, BUFGCE, BUFGCE_DIV, BUFGCTRL, IOB, MMCME3_ADV, PLLE3_ADV, PLL_SELECT_SITE, RIU_OR, etc. The adjacent 60 CLBs and the shared interconnect.

Automatic Adjustments for PU on PBlocks

In UltraScale devices, there is no height requirement of Pblocks for RESET_AFTER_RECONFIG capability. For this reason, the feature is always ON, and there are no special requirements that need to be met. However, to ensure that the Pblock does not violate any rules for minimum PU sizes, the SNAPPING_MODE property is also always on by default, and automatically adjusts the Pblock to make sure it is valid for PR.

Figure 6-1 and Figure 6-2, page 77 below give an example of how SNAPPING_MODE adjusts the Pblock for PU alignment. In Figure 6-1, despite the larger outer rectangle, only the selected tiles belong to the RP Pblock. The upper block RAM and DSP sites are not included because they are not fully contained in the Pblock, and the associated CLB sites are not included either, based on the PU rules. There are also CLB sites on both the left and right edge that are not included in the Pblock because the adjacent CLBs are not owned by the original rectangle.

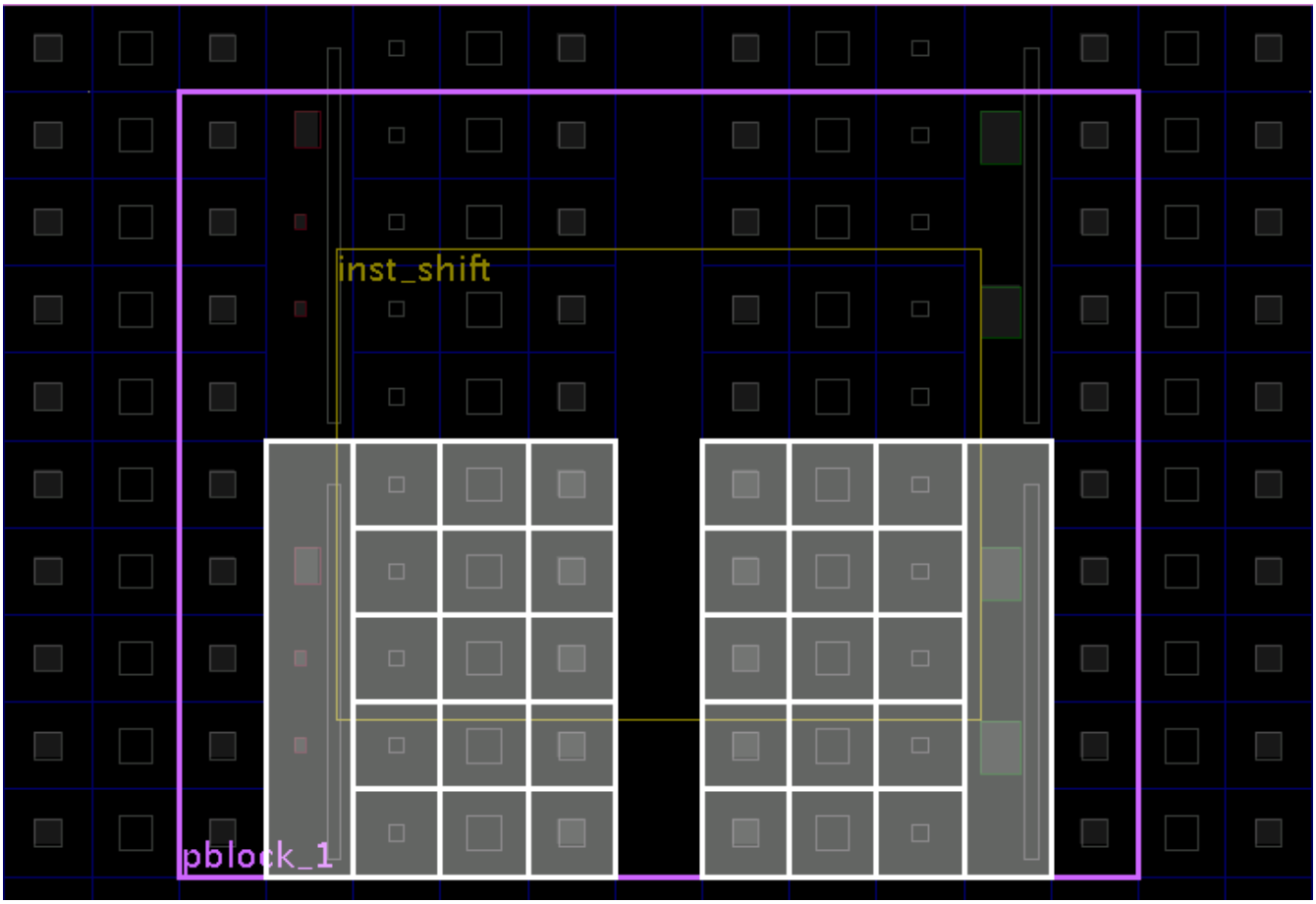

Figure 6-1: **SNAPPING_MODE Example - UltraScale**

While SNAPPING_MODE made the above Pblock legal for the RP, it is possible that the intent was to include all of these sites. By making a small adjustment to the original Pblock rectangle, you can prevent SNAPPING_MODE from removing sites that are intended for the PR region. In Figure 6-2 the Pblock has been expanded by one CLB on the left, right, and top edges. The highlighted tiles that are owned by the RP Pblock now match the outer rectangle.

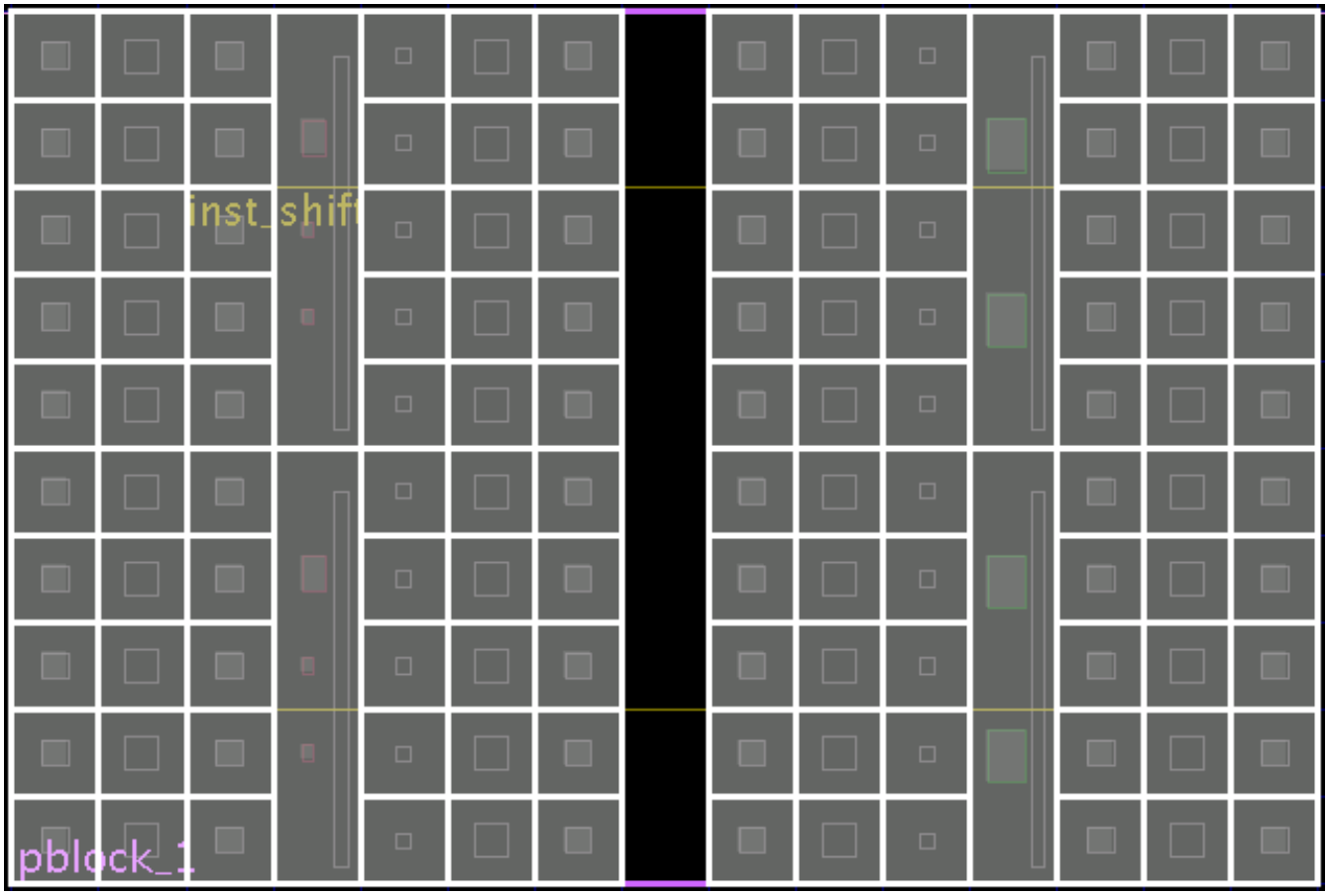

Figure 6-2: **PU Aligned Pblock**

Note: The images Figure 6-1 and Figure 6-2 were created using highlighting scripts that the Vivado Design Suite tools create automatically created when the parameter HD.VISUAL is set in. The following steps can be used to reproduce these images for debugging/verifying Pblocks:

1. In the Vivado IDE Tcl Console, type:

   ```
   set_param hd.visual 1
   ```

2. Create or make an adjustment to a Pblock.
3. Source the highlighting script that was generated by the Vivado tools.

   ```
   source ./hd_visual/<pblock_name>_AllTiles.tcl
   ```

IMPORTANT: *The `SNAPPING_MODE` property for UltraScale devices is always set to `ON` and should not be adjusted or turned `OFF`. `SNAPPING_MODE` has special behavior for certain types of logic like IOB and Laguna for SSI devices. In these cases, `SNAPPING_MODE` expands the PR Pblock outward to include all necessary resources.*

Sharing Configuration Frames between RP and Static Logic

Even though UltraScale device Pblocks are not required to be frame aligned (that is, the height of the clock region), Partial Reconfiguration still programs the entire configuration frame. This means that logic outside of the RP is overwritten. This does not cause any issues in PR, but in previous architectures there were some limitations about what kind of static logic could be in the same frame as reconfigurable logic.

For UltraScale devices, any static logic can be placed in the same configuration frame as the RM, in any sites not owned by the RP Pblock. This includes block RAM, DSP, and LUT RAM. There is still a restriction however, that there can be only one RP per configuration frame. That means you cannot vertically stack two RPs in the same clock region.

Global Clocking Rules

As with architectures previous to UltraScale, all unique clocks driving the Reconfigurable Partition are pre-routed to every clock region in which the RP owns sites. Effectively, this means that the total number of global clocks driving the Reconfigurable Partition (regardless of size) is a maximum of 24. Higher clock utilization is possible when the clock source is in the RM, since these do not need to be pre-routed to every clock region. For this reason it is always important to carefully consider the RP Pblock size and shape. However, one difference in the UltraScale architecture is that there are now 24 global clocks available per clock region instead of the 12 available in 7 series devices.

Note: For BUFGCTRL components, the `PRESELECT_I0` and `PRESELECT_I1` properties are ignored during partial reconfiguration, even with `RESET_AFTER_RECONFIG` enabled. The clock source selected depends only on the select and clock enable inputs of the `BUFGCTRL` instance.

Currently, an RM is not allowed to drive a clock out of the module. A clock created in the static region can drive an input pin of an RP, and clocks created inside an RM can only drive logic within that RM. However, a clock net cannot drive an output pin of an RP. If this case is detected, a DRC error is issued.

I/O Rules

In UltraScale devices, I/O logic and buffers can be included in an RP. While the I/O can be modified from one RM to another, there are some rules that must be followed.

The following checks are done between all configurations that use the I/O sites. If an I/O site changes from being used to unused, or vice versa, then these checks are not done for those configurations.

- The I/O direction, standard, reference voltage, slew, and drive strength must be the same between all RMs whenever the I/O is used.
- For `DCI_CASCADE`, the member bank assignments between RMs cannot overlap.
  - Legal example: In Configuration 1, `DCI_CASCADE` has banks 12, 13. In Configuration 2, `DCI_CASCADE` has banks 14, 15 and 16. They do not have overlapped banks.
  - Illegal example: In Configuration 1, `DCI_CASCADE` has banks 12 and 13. In Configuration 2, `DCI_CASCADE` has banks 13, 14, 15 and 16. In this case bank 13 overlaps.
- For `DCI_CASCADE`, member banks must be fully contained within the reconfigurable region. All of the member banks for the same `DCI_CASCADE` must be in either the same RP Pblock, or completely in static.

Changes to the IOB from one configuration to another are limited by the rules above. However, adding the I/O sites into the RP requires that the entire PU (encompassing the I/O bank, BITSLICE, MMCM, PLL, and one column of CLBs plus shared interconnect) be added. All components in this fundamental region are reconfigured and reinitialized, and adding these other site types to the reconfigurable region can be beneficial in some cases for these reasons:

- Adding I/O sites allows use of the routing resources of the I/O, which reduces congestion (instead of increasing congestion, as it could if the I/O sites were in Static, and caused a gap in the reconfigurable region).
- Allows reconfiguration of other clocking resources like the MMCM and PLL.
- Allows reconfiguration of other I/O logic sites such as BITSLICE and BITSLICE_CONTROL.

Regardless of whether or not the I/O usage or characteristics change during reconfiguration, the entire bank is reconfigured. During reconfiguration, all I/O in the banks defined by the RP pblock is held with the dedicated global tri-state (GTS) signal, which is released at the end of reconfiguration.

Using High Speed Transceivers

Xilinx high speed transceivers (GTH, GTY) are supported within a Reconfigurable Partition. As with other reconfigurable site types, the entire PU must be included. For the UltraScale GT transceivers, the PU includes

- 4 GT_CHANNEL sites (GT Quad)
- Associated GT_COMMON site
- Associated BUFG_GT_SYNC sites
- Associated BUFG_GT sites
- Associated Interconnect and CLB sites

The required GT PU is the entire height of a clock region. As with previous architectures, it is also possible to leave the GT components in static logic and change the functionality through the DRP. For more information on using UltraScale transceivers, see the *UltraScale Architecture GTH Transceivers User Gui*de (UG576) [Ref 24] or the *UltraScale Architecture GTY Transceivers User Gui*de (UG578) [Ref 25].

Partial Reconfiguration Checklist for UltraScale Device Designs

Xilinx highly encourages the following for an UltraScale device design using Partial Reconfiguration:

Recommended Clocking Networks

Are you using Global Clock Buffers or Clock Modifying Blocks (MMCM, PLL)?

These blocks can be reconfigured, but all elements in this frame type must be reconfigured. This includes an entire I/O bank and all clocking elements in that shared region, plus one column of CLBs that share the interconnect.

See Design Elements Inside Reconfigurable Modules, page 74 for more information, and Global Clocking Rules, page 78 for complete details on global clock implementation.

In addition, the following restrictions are currently enforced by Vivado Design Suite DRC rules. The use of clocking resources BUFGCTRL, BUFG_CE and BUFG_GT is supported with the following restrictions:

- Xilinx recommends using rectangular Pblock shapes. Non-rectangular shapes are also supported for RPs with clocking logic, as long as the tallest column of the Pblock is aligned vertically and horizontally with the clock region. The tallest column of the RP Pblock must also range the IOB, and this range must cover the full height of all the rectangles that define the RP Pblock, as shown in Figure 6-3, page 81. In other words, this vertical column of IOB ranges must be able to access all rows of the Pblock. Pblock shapes like a sideways "L" are not supported unless the vertical section of the shaped includes the IOB range.

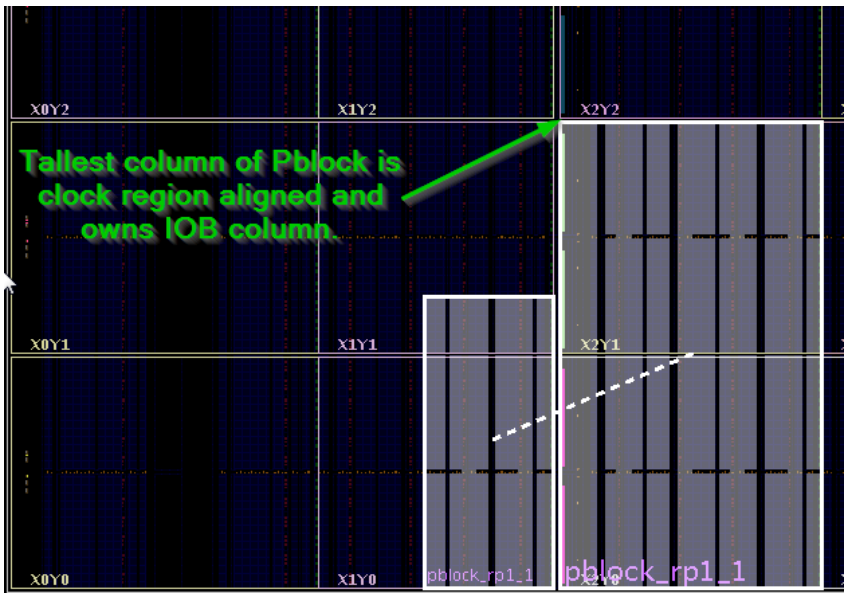

Figure 6-3: **Tallest Column of Pblock Clock Region Aligned**

- A gap is defined as an unranged site type with ranged sites on both sides of it. The following gaps are not allowed:
  - Gaps in the IOB/XIPHY ranges, such as the gap in the IOB column shown in Figure 6-4 below.

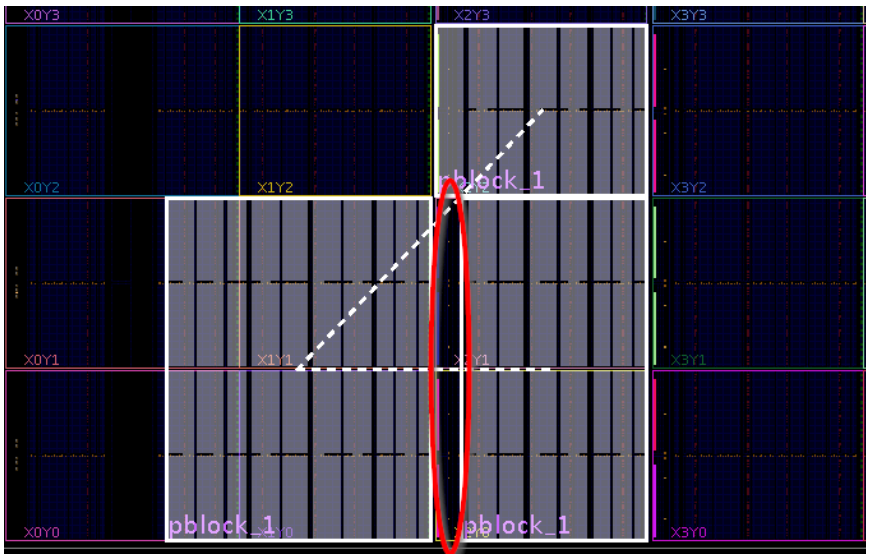

Figure 6-4: **Gap in the IOB Column Not Supported**

  - Gaps in the DSP ranges.
- A clock region cannot be shared by two RP Pblocks if:
  - At least one of them has a global clock source.
  - The other has ranged a global clock source.
- For SSI Technology devices, clock regions in the corners of an SLR cannot be included in the RP region if the following are true:
  - The pblock is contained in single SLR.
  - The RMs contains clocking logic (MMCM, PLL, BUFGCTRL, BUFG_CE, or BUFG_GT).

These rules are enforced by the following list of DRC checks:

DRC	Description
HDPR-57	Reconfigurable Pblock using Global Clock resources must share a common clock region column that does not contain unused LAGUNA sites.
HDPR-58	Reconfigurable Pblock using Global Clock resources must share a common clock region column that does not contain CONFIG_SITES.
HDPR-59	Clock Net Rule Violation
HDPR-60	Reconfigurable Pblock using Global Clock resources must have contiguous clock regions.
HDPR-61	Reconfigurable Pblock using Global Clock resources must not skip over clock region rows.
HDPR-62	Reconfigurable Pblock using Global Clock resources must share a common clock region column.
HDPR-63	Reconfigurable Pblock using Global Clock resources must have complete horizontal spines through edge DSP columns.
HDPR-64	Reconfigurable Pblock using Global Clock resources must have complete horizontal spines through XIPHY tiles.
HDPR-65	Reconfigurable Pblock using Global Clock resources cannot share Clock Regions.

Configuration Feature Blocks

Are you using device feature blocks (BSCAN, DCIRESET, FRAME_ECC, ICAP, STARTUP, USR_ACCESS)?

These featured blocks must be in static logic.

See Design Elements Inside Reconfigurable Modules, page 74 for more information.

SSI Technology

Does the Pblock span an SLR of an SSI device?

If using an SSI device it is recommended to keep a PR region within a single SLR. However, for UltraScale devices, if a PR Pblock must span an SLR, the necessary Laguna sites must be included to allow for routing across this boundary. This requires that at least one full clock regions belongs to the PR region on both sides of the SLR boundary. SNAPPING_MODE automatically expands the Pblock to own the necessary sites as long as the Laguna resource is ranged. Verify this site type is selected when defining the PR Pblock.

For more information on SSI Technology devices and Laguna, see Devices using Stacked Silicon Interconnect (SSI) Technology in the *UltraScale Architecture Configurable Logic Block User Guide* (UG574) [Ref 29].

High Speed Transceiver Blocks

Do you have high speed transceivers in your design?

High speed transceivers can be reconfigured. An entire quad, including all component types (GT_CHANNEL, GT_COMMON, BUFG_GT) must be reconfigured together.

See Using High Speed Transceivers, page 80 for specific requirements.

System Generator DSP Cores, HLS cores, or IP Integrator Block Diagrams

Are you using System Generator DSP cores, HLS cores, or IP Integrator block diagrams in your Partial Reconfiguration design?

Any type of source can be used as long as it follows the fundamental requirements for Partial Reconfiguration. Any code processed by SysGen, HLS, or IP Integrator (or other tools) is eventually synthesized. The resulting design checkpoint or netlist must be comprised entirely of reconfigurable elements in order for it to be legally included in an RP.

Packing I/Os into Reconfigurable Partitions

Do you have I/Os in reconfigurable modules?

I/Os can be partially reconfigured. An entire I/O bank, along with all I/O logic (XiPhy) and clocking resources, must be reconfigured at once.

See Design Elements Inside Reconfigurable Modules, page 74 for more information.

Packing Logic into Reconfigurable Partitions

Is all logic that must be packed together in the same Reconfigurable Partition?

Any logic that must be packed together must be in the same RP and RM.

See Packing Logic, page 48 for more information.

Packing Critical Paths into Reconfigurable Partitions

Are critical paths contained within the same partition?

Reconfigurable partition boundaries limit some optimization and packing, so critical paths should be contained within the same partition.

See Packing Logic, page 48 for more information.

Floorplanning

Can your Reconfigurable Partitions be floorplanned efficiently?

See Creating Pblocks for UltraScale Devices, page 75 for more information.

Recommended Decoupling Logic

Have you created decoupling logic on the outputs of your RMs?

During reconfiguration the outputs of RPs are in an indeterminate state, so decoupling logic must be used to prevent static data corruption.

See Decoupling Functionality, page 52 for more information.

Recommended Reset After Reconfiguration

Are you resetting the logic in an RM after reconfiguration?

Reset After Reconfiguration is always enabled for UltraScale devices.

See Apply Reset After Reconfiguration, page 38 for more information.

Debugging with Logic Analyzer Blocks

Are you using the Vivado Logic Analyzer with your Partial Reconfiguration design?

Vivado logic analyzer (ILA/VIO debug cores) can be used in your Partial Reconfiguration design, but they must be in static logic.

Efficient Reconfigurable Partition Pblocks

Have you created efficient Reconfigurable Partition Pblock(s) for your design?

A Reconfigurable Partition Pblock can be any height, but multiple Reconfigurable Partitions cannot be stacked vertically within a single clock region.

See Creating Pblocks for UltraScale Devices, page 75 for more information.

Validating Configurations

How do you validate consistency between configurations?

The `pr_verify` command is used to make sure all configurations have matching imported resources.

See Verifying Configurations in Chapter 3 for more information.

Configuration Requirements

Are you aware of the particular configuration requirements for Partial Reconfiguration for your design and device?

Each device family has specific configuration requirements and considerations.

See the Chapter 7, Configuring the Device for more information.

Configuring the Device

Overview

This chapter describes the system design considerations when configuring your device with a partial BIT file, as well as architectural features in the FPGA that facilitate Partial Reconfiguration. Because most aspects of Partial Reconfiguration are no different than standard full configuration, this section concentrates on the details that are unique to PR.

Any of the following configuration ports can be used to load the partial bitstream: SelectMAP, Serial, JTAG, or ICAP (Internal Configuration Access Port). For Zynq®-7000 AP SoC devices, deliver partial bitstreams using the JTAG or PCAP (Processor Configuration Access Port) ports. For UltraScale™ devices, the MCAP (Media Configuration Access Port) within the PCIe® block is also a valid configuration port.

To use SelectMAP or Serial modes for loading a partial BIT file, these pins must be reserved for use after the initial device configuration. This is achieved by using the `BITSTREAM.CONFIG.PERSIST` property to keep the dual-purpose I/O for configuration usage and to set the configuration width. Refer to this link in the *Vivado Design Suite User Guide: Programming and Debugging* (UG908) [Ref 27]. The Tcl command syntax to set this property is:

```
set_property BITSTREAM.CONFIG.PERSIST <value> [current_design]
```

where `<value>` is either `No` or `Yes`.

Partial bitstreams contain all the configuration commands and data necessary for Partial Reconfiguration. The task of loading a partial bitstream into an FPGA does not require knowledge of the physical location of the RM because configuration frame addressing information is included in the partial bitstream. A valid partial bitstream cannot be sent to the wrong part of the FPGA.

A Partial Reconfiguration controller retrieves the partial bitstream from memory, then delivers it to a configuration port. The Partial Reconfiguration control logic can either reside in an external device (for example, a processor) or in the programmable logic of the FPGA to be reconfigured. A user-designed internal PR controller loads partial bitstreams through the ICAP interface. As with any other logic in the static design, the internal Partial Reconfiguration control circuitry operates without interruption throughout the Partial Reconfiguration process.

Internal configuration can consist of either a custom state machine, or an embedded processor such as MicroBlaze™. For a Zynq-7000 AP SoC, the Processor Subsystem (PS) can be used to manage Partial Reconfiguration events.

Note: For Zynq-7000 AP SoC devices, the Programmable Logic (PL) can be partially reconfigured, but the Processing System cannot.

As an aid in debugging Partial Reconfiguration designs and PR control logic, the Vivado® Logic Analyzer can be used to load full and partial bitstreams into an FPGA by means of the JTAG port.

For more information on loading a bitstream into the configuration ports, see the Configuration Interfaces chapter in these documents:

- *7 Series FPGAs Configuration User Guide* (UG470) [Ref 7]
- *Zynq-7000 AP SoC Technical Reference Manual* (UG585) [Ref 9]

Configuration Modes

Partial Reconfiguration is supported using the following configuration modes:

- **ICAP:** A good choice for user configuration solutions. Requires the creation of an ICAP controller as well as logic to drive the ICAP interface.
- **MCAP:** (UltraScale devices only) Provides a dedicated connection to the ICAP from one specific PCIe® block per device.
- **PCAP:** The primary configuration mechanism for Zynq-7000 AP SoC designs.
- **JTAG:** A good interface for quick testing or debug. Can be driven with the Vivado Logic Analyzer.
- **Slave SelectMAP** or **Slave Serial:** A good choice to perform full configuration and Partial Reconfiguration over the same interface.

Master modes are not directly supported because IPROG housecleaning clears the configuration memory.

Bitstream Type Definitions

When designs are compiled for Partial Reconfiguration in Xilinx devices, different types of bitstreams are created. This section defines terminology and explains the details for each type of bitstream for 7 series and UltraScale devices. The types of bitstreams are:

- Full Configuration Bitstreams
- Partial Bitstreams
- Blanking Bitstreams
- Clearing Bitstreams

Full Configuration Bitstreams

All PR designs start with standard configuration of the full device using a full configuration bitstream. The format and structure is no different than for a flat design solution, and there is no difference in how this bitstream can be used to initially program the FPGA. However, note that the design itself has been processed in preparation for partial reconfiguration of the device after the full programming has been done. All standard features, such as encryption and compression, are supported.

Reconfigurable Partitions (RP) set as black boxes are supported, so Reconfigurable Modules (RM) with no functionality can be delivered as part of the initial configuration, to be replaced later with a desired Reconfigurable Module. Bitstream compression can be effective in this case, reducing bitstream size and initial configuration time.

Downloading a Full BIT File

The FPGA in a digital system is configured after power on reset by downloading a full BIT file, either directly from a PROM or from a general purpose memory space by a microprocessor. A full BIT file contains all the information necessary to reset the FPGA, configure it with a complete design, and verify that the BIT file is not corrupt. The figure below illustrates this process.

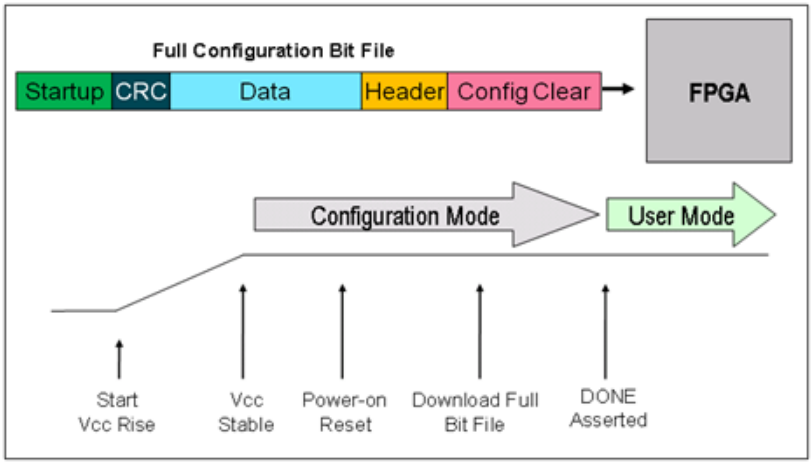

Figure 7-1: **Configuring with a Full BIT File**

After the initial configuration is completed and verified, the FPGA enters user mode, and the downloaded design begins functioning. If a corrupt BIT file is detected, the DONE signal is never asserted, the FPGA never enters user mode, and the corrupt design never starts functioning.

Partial Bitstreams

Partial bitstreams are delivered during normal device operation to replace functionality in a pre-defined device region. These bitstreams have the same structure as full bitstreams but are limited to specific address sets to program a specific portion of the device. Dedicated PR features such as per-frame CRC checks (to ensure bitstream integrity) and automatic initialization (so the region starts in a known state) are available, along with full bitstream features such as encryption and compression.

The size of a partial bitstream is directly proportional to the size of the region it is reconfiguring. For example, if the Reconfigurable Partition is composed of 20% of the device resources, the partial bitstream is roughly 20% the size of the full design bitstream.

Partial bitstreams are fully self-contained, so they are delivered to an appropriate configuration port. All addressing, header, and footer details are contained within these bitstreams, just as they would be for full configuration bitstreams. You deliver partial bitstreams are delivered to the FPGA through any external non-master configuration mode, such as JTAG, Slave Serial, or Slave SelectMap. Internal configuration access includes the ICAP (all devices), PCAP (Zynq-7000 AP SoC devices), and MCAP (UltraScale devices through PCIe).

Partial bitstreams are automatically created when `write_bitstream` is run on a PR configuration. Each partial bitstream file name references your top-level design name, plus the pblock name for the Reconfigurable Partition, `plus _partial`. For example, for a full design bit file `top_first.bit`, a partial bit file could be named `top_first_pblock_red_partial.bit`.

RECOMMENDED: *The pblock instance is always the same, regardless of the RM contained within, so it is recommended that you use a descriptive base configuration name or rename the partial bit files to clarify which module it represents.*

Downloading a Partial BIT File

A partially reconfigured FPGA is in user mode while the partial BIT file is loaded. This allows the portion of the FPGA logic not being reconfigured to continue functioning while the reconfigurable portion is modified. Figure 7-2 illustrates this process.

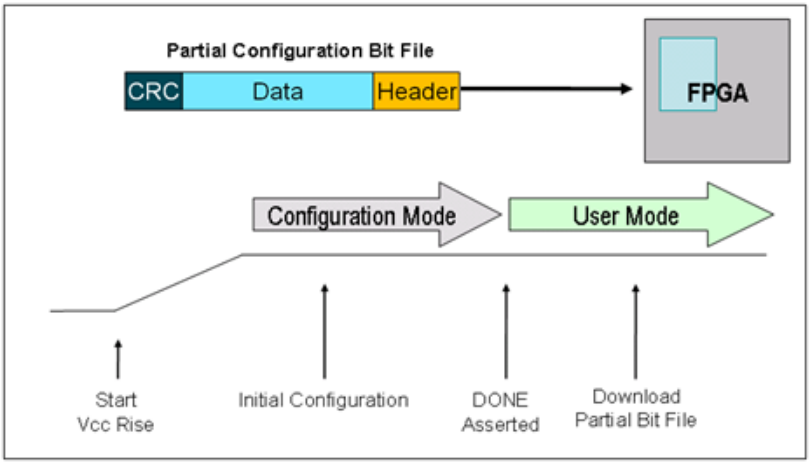

Figure 7-2: **Configuring with a Partial BIT File**

The partial BIT file has a simplified header, and there is no startup sequence that brings the FPGA into user mode. The BIT file contains (essentially, and with default settings) only frame address and configuration data, plus a final checksum value. Additional CRC checks can be inserted, if desired, to perform bitstream integrity checking.

If Reset After Reconfiguration is used, the DONE pin pulls LOW when reconfiguration begins and pulls HIGH again when partial reconfiguration successfully completes, although the partial bitstream can still be monitored internally as well. In UltraScale devices, this behavior is echoed on the PRDONE output pin of the ICAP.

Note: With UltraScale devices, the DONE and PRDONE pins pull LOW at the beginning of the clearing bitstream and remain low until the end of the partial bitstream because the two bitstreams together constitute a complete partial reconfiguration sequence. The DONE/PRDONE pin does NOT return high at the end of the clearing bitstream.

If Reset After Reconfiguration is not selected, you must monitor the data being sent to know when configuration has completed. The end of a partial BIT file has a DESYNCH word (0000000D) that informs the configuration engine that the BIT file has been completely delivered. This word is given after a series of padding NO OP commands, ensuring that once the DESYNCH has been reached, all the configuration data has already been sent to the target frames throughout the device. As soon as the complete partial BIT file has been sent to the configuration port, it is safe to release the reconfiguration region for active use.

Blanking Bitstreams

A blanking bitstream is a specific type of partial bitstream, one that represents a black box. It removes the functionality of an existing Reconfigurable Module by replacing it with new functionality, which consists simply of tie-off LUTs on all appropriate module I/O.

To create a black box Reconfigurable Module, you remove the logical and physical representation of a fully placed and routed design configuration and replaces it with tie-off LUTs. Starting with a routed configuration (with the static design locked) in active memory, run these steps:

```
update_design -cell <foo> -black_box
update_design -cell <foo> -buffer_ports
place_design
route_design
```

The design must be placed and routed to implement the LUTs that have been inserted into the design. Outputs of the black box RM are tied to ground by default, but can be set to Vcc by setting the HD.PARTPIN_TIEOFF on desired ports.

Compression can be used to greatly reduce the size of blanking bitstreams. Note that these bitstreams still contain, not only the tie-off LUTs, but also any static routing that happens to pass through this region of the FPGA. Blanking bitstreams are generated and named in the same way as standard partial bitstreams, as the black box variation is saved as another configuration checkpoint.

Clearing Bitstreams

Unlike the bitstream types noted above, this type is for UltraScale devices only. A new requirement for this architecture is to "clear" an existing module before loading a new module. This clearing bitstream prepares the device for the delivery of any subsequent partial bitstream for that Reconfigurable Partition by establishing the global signal mask for the region to be reconfigured. Although the existing module is technically not removed (the current logical module remains), it is easiest to think of it this way. If a clearing bitstream is not delivered, the subsequent reconfigurable module will not be initialized.

Clearing bitstreams are *not* partial bitstreams. They comprise less than 10% of the frames for the target region and are therefore less than 10% the size of the corresponding partial bitstreams. They do not change the functionality but shut down clocks driving logic in the region. They must be delivered between partial bitstreams and should always be followed as soon as possible by the next partial bitstream.

Each clearing bitstream is built for a specific Reconfigurable Module and must be applied after that module has been used, and must be sent to the configuration engine immediately before the next partial bitstream is delivered. For example, to transition from module A to module B, the clearing bitstream for A must be delivered just before the partial bitstream for B is delivered. To transition from module B back to module A, the clearing bitstream for B must be delivered just before the partial bitstream for A is delivered. This is the case even if any partial bitstream in question is a blanking bitstream.

Clearing bitstreams are automatically generated and have the same name as partial bitstreams with `_clear` at the end. Looking at the example above, if `top_first` is an UltraScale device design, the clearing bit file name would be `top_first_pblock_red_partial_clear.bit`.

Partial Reconfiguration through ICAP for Zynq Devices

The primary configuration mechanism for the programmable logic (PL) of Zynq devices is through the processing system (PS), which delivers bitstreams to the PCAP. The most straightforward mechanism for partial reconfiguration is also via this path. However, to manage partial reconfiguration completely within the PL (either through the PR Controller IP or through a custom-designed controller module), partial bitstreams can also be delivered to the ICAP, just as they can be for FPGA devices.

The PCAP and ICAP interfaces are mutually exclusive and cannot be used simultaneously. Switching between ICAP and PCAP is possible, but you must ensure that no commands or data are being transmitted or received before changing interfaces. Failure to do this could lead to unexpected behavior. Bit 27 `(PCAP_PR)` of the Control Register `(devc.CTRL)` selects between ICAP and PCAP for PL reconfiguration. The default is `PCAP (1)`, but that can be changed to `ICAP (0)` to enable this configuration port. Note that bit 28 `(PCAP_MODE)` must also be set to 1, which is the default. For more details, see the *Zynq-7000 All Programmable SoC Technical Reference Manual* (UG585) [Ref 9].

Accessing the Configuration Engine through the MCAP

UltraScale devices introduce a dedicated connection from one specific PCIe block on a device to the configuration engine, providing an efficient mechanism for delivering partial bitstreams. No explicit routes are required to connect the PCIe block to the ICAP, saving considerable resources.

To enable this capability, select the **PR over PCIe** value for the **Tandem Configuration or Partial Reconfiguration** option (as shown in the figure below) when generating the UltraScale FPGA Gen3 Integrated Block for PCI Express IP. Advanced Mode must be selected, the MCAP-enabled PCIe Block Location must be selected, and a device that currently supports Partial Reconfiguration and Tandem Configuration must be selected before the option becomes available.

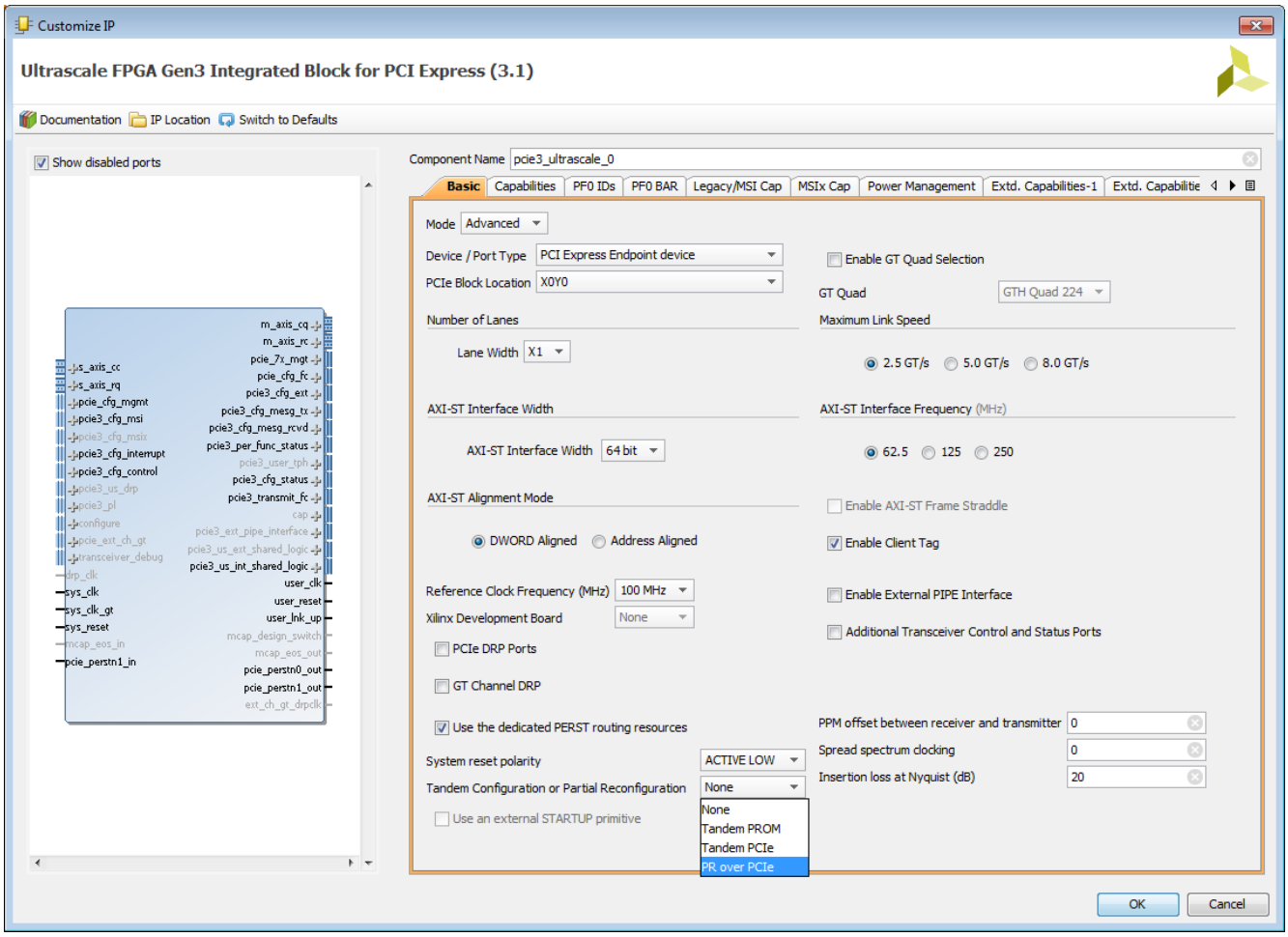

Figure 7-3: **Selecting the PR over PCIe Option when Generating the Xilinx PCIe IP**

The PCIe block that must be selected in most cases is the lowest instance in the device, except for SSI devices with three Super Logic Regions (SLRs), in which case it is the lowest PCIe instance in the center SLR. A complete listing of the specific supported blocks is shown here. All other PCIe blocks do not have the dedicated MCAP feature.

Table 7-1: **PCIe Block and Reset Locations Supporting PR, by Device**

Device	PCIe Block Instance Supporting Partial Reconfiguration	PCIe Reset Location
Kintex® UltraScale		
XCKU025*	PCIE_3_1_X0Y0	IOB_X1Y103
XCKU035	PCIE_3_1_X0Y0	IOB_X1Y103
XCKU040	PCIE_3_1_X0Y0	IOB_X1Y103
XCKU060	PCIE_3_1_X0Y0	IOB_X2Y103
XCKU085	PCIE_3_1_X0Y0	IOB_X2Y103
XCKU095	PCIE_3_1_X0Y0	IOB_X1Y103
XCKU115	PCIE_3_1_X0Y0	IOB_X2Y103
Virtex® UltraScale		
XCVU065	PCIE_3_1_X0Y0	IOB_X1Y103
XCVU080	PCIE_3_1_X0Y0	IOB_X1Y103
XCVU095	PCIE_3_1_X0Y0	IOB_X1Y103
XCVU125	PCIE_3_1_X0Y0	IOB_X1Y103
XCVU160	PCIE_3_1_X0Y1	IOB_X1Y363
XCVU190	PCIE_3_1_X0Y2	IOB_X1Y363
XCVU440*	PCIE_3_1_X0Y2	IOB_X1Y363

*: Not yet supported in Vivado software.

The MCAP is capable of operating at 200 MHz with a 32-bit data path. Traditionally bitstreams are loaded into the MCAP from a host PC through PCI Express configuration packets. In these systems the host PC and host PC software are the main factors which limit MCAP performance and bitstream throughput. Because PCIe performance of specific host PC and host PC software can vary widely, overall MCAP performance throughput might vary.

For more information and sample drivers, see the answer record, *Bitstream Loading across the PCI Express Link in UltraScale Devices for Tandem PCIe and Partial Reconfiguration* (AR# 64761) [Ref 5].

Formatting BIN Files for Delivery to Internal Configuration Ports

Partial bit files have the same basic format as full bit files, but they are reduced to the set of configuration frames for the target region and restricted to the set of events that make sense for active devices. Partial bit files can be:

- Delivered to external interfaces, such as JTAG or slave configuration ports.
- Reformatted as BIN files to be delivered to the internal configuration ports: ICAP (7 series or UltraScale devices), PCAP (Zynq devices only) or MCAP (UltraScale devices only).

Generate BIN files using the `write_cfgmem` utility. Three options are critical:

- Set `-format` as BIN to generate that file type.
- Use `-interface` to select the SelectMap width, and use `SMAPx32` for PCAP or `MCAP` for UltraScale ICAP.
  - `SMAPx16` and `SMAPx8` (default) can also be used for the 7 series ICAP.
  - `SMAPx8` is required for 7 series encrypted partial bitstreams.
- You must use `-disablebitswap` to target the PCAP or MCAP.

Examples

ICAP (for 7 series devices)

```
write_cfgmem -format BIN -interface SMAPx8 -loadbit "up 0x0 <partial_bitfile>”
```

ICAP (for UltraScale devices)

```
write_cfgmem -format BIN -interface SMAPx32 -loadbit "up 0x0 <partial_bitfile>”
```

PCAP (for Zynq-7000 SoC devices) or MCAP (for one specific PCIe block per UltraScale device)

```
write_cfgmem -format BIN -interface SMAPx32 -disablebitswap -loadbit "up 0x0
<partial_bitfile>”
```

Summary of BIT Files for UltraScale Devices

With the finer granularity of global signals (that is, GSR) and the ability to reconfigure new element types, a new configuration process is necessary. Prior to loading in a partial bitstream for a new Reconfigurable Module, the existing Reconfigurable Module must be cleared. This clearing bitstream prepares the device for delivery of any subsequent partial bitstream for that Reconfigurable Partition by establishing the global signal mask for the region to be reconfigured. Although the existing module technically is not removed, it is easiest to think of it this way.

When running `write_bitstream` on a design configuration with Reconfigurable Partitions, a clearing BIT file per RP is created. For example, take a design in which two Reconfigurable Partitions (RP1 and RP2), with two Reconfigurable Modules each, A1 and B1 into RP1, and A2 and B2 into RP2, have been implemented. Two configurations (configA and configB) have been run through place and route, and `pr_verify` has passed. When bitstreams are generated, each configuration produces five bitstreams. For configA, these could be named:

- `configA.bit` - This is the full design bitstream that is used to configure the device from power-up. This contains the static design plus functions A1 and A2.
- `configA_RP1_A1_partial.bit` - This is the partial BIT file for function A1. This is loaded after another RM has been cleared from this Reconfigurable Partition.
- `configA_RP1_A1_partial_clear.bit` - This is the clearing BIT file for function A1. Before loading in any other partial BIT file into RP1 *after function* A1, this file must be loaded.
- `configA_RP2_A2_partial.bit` - This is the partial BIT file for function A2. This is loaded after another RM has been cleared from this Reconfigurable Partition.
- `configA_RP2_A2_partial_clear.bit` - This is the clearing BIT file for function A2. Before loading in any other partial BIT file into RP2 *after function* A2, this file must be loaded.

Likewise, configB produces five similar bitstreams:

- `configB.bit` - This is the full design bitstream that is used to configure the device from power-up. This contains the static design plus functions B1 and B2.
- `configB_RP1_B1_partial.bit` - This is the partial BIT file for function B1. This is loaded after another RM has been cleared from this Reconfigurable Partition.
- `configB_RP1_B1_partial_clear.bit` - This is the clearing BIT file for function B1. Before loading in any other partial BIT file into RP1 *after function* B1, this file must be loaded.
- `configB_RP2_B2_partial.bit` - This is the partial BIT file for function B2. This is loaded after another RM has been cleared from this Reconfigurable Partition.

- `configB_RP2_B2_partial_clear.bit` - This is the clearing BIT file for function B2. Before loading in any other partial BIT file into RP2 *after function* B2, this file must be loaded.

The sequence for any reconfiguration is to first load a clearing BIT file for a current Reconfigurable Module, immediately followed by a new Reconfigurable Module. For example, to transition Reconfigurable Partition RP1 from function A1 to function B1, first load the BIT file `configA_RP1_A1_partial_clear.bit`, then load `configB_RP1_B1_partial.bit`. The first bitstream prepares the region by opening the mask, and the second bitstream loads the new function, initializes only that region, then closes the mask.

If a clearing bit file is not loaded, initialization routines (GSR) have no effect. If a clearing file for a different Reconfigurable Partition is loaded, then that RP is initialized instead of the one that has been just reconfigured. If the incorrect clearing file for the proper RP is used, the current RM or possibly even the static design could be disrupted until the following partial bit file has been loaded.

System Design for Configuring an FPGA

A partial BIT file can be downloaded to the FPGA in the same manner as a full BIT file. An external microprocessor determines which partial BIT file should be downloaded, where it exists in an external memory space, and directs the partial BIT file to a standard FPGA configuration port such as JTAG, Select MAP or serial interface. The FPGA processes the partial BIT file correctly without any special instruction that it is receiving a partial BIT file.

It is common to assert the INIT or PROG signals on the FPGA configuration interface before downloading a full BIT file. This must not be done before downloading a partial BIT file, as that would indicate the delivery of a full BIT file, not a partial one.

Any indication to the working design that a partial BIT file will be sent (such as holding enable signals and disabling clocks) must be done in the design—and not by means of dedicated FPGA configuration pins. Figure 7-4, page 99 shows the process of configuring through a microprocessor.

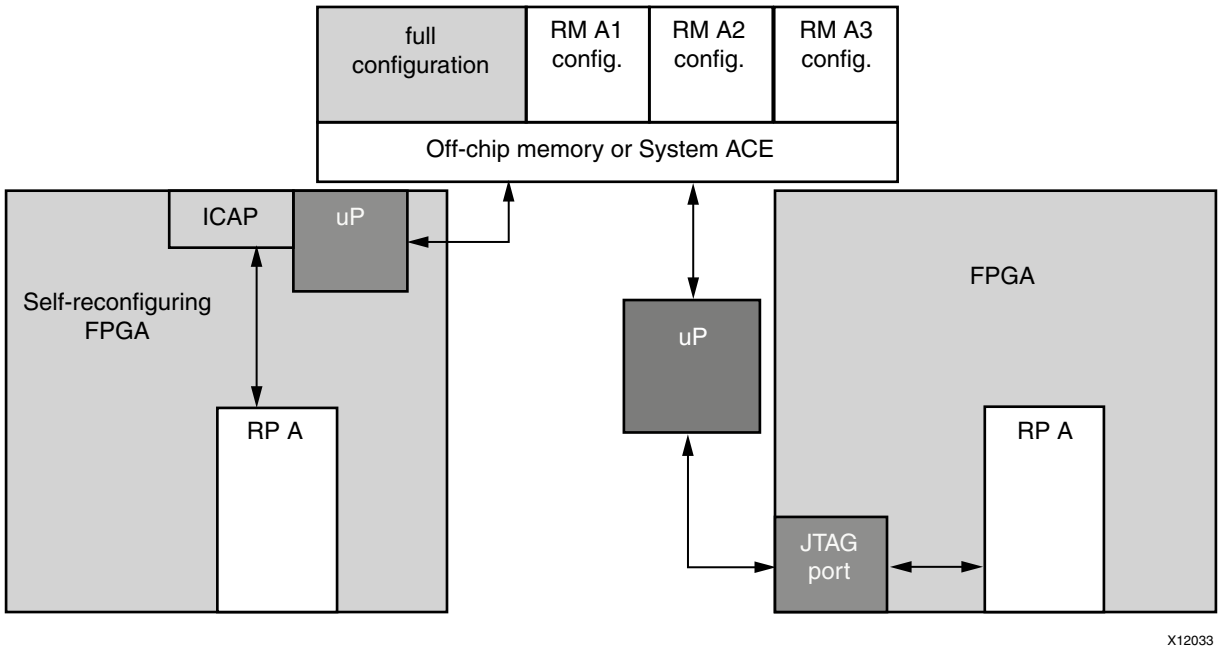

Figure 7-4: **Configuring Through a Microprocessor**

In addition to the standard configuration interfaces, Partial Reconfiguration supports configuration by means of the Internal Configuration Access Port (ICAP). The ICAP protocol is identical to SelectMAP and is described in the Configuration User Guide for the target device. The ICAP library primitive can be instantiated in the HDL description of the FPGA design, thus enabling analysis and control of the partial BIT file before it is sent to the configuration port. The partial BIT file can be downloaded to the FPGA through general purpose I/O or gigabit transceivers and then routed to the ICAP in the FPGA programmable logic.

The ICAP must be used with an 8-bit bus only for Partial Reconfiguration for encrypted 7 series BIT files. Reconfiguration through external configuration ports is permitted only when bitstream readback security is not set to Level2.

Partial BIT File Integrity

Error detection and recovery of partial BIT files have unique requirements compared to loading a full BIT file. If an error is detected in a full BIT file when it is being loaded into an FPGA, the FPGA never enters user mode. The error is detected after the corrupt design has been loaded into configuration memory, and specific signals are asserted to indicate an error condition. Because the FPGA never enters user mode, the corrupt design never becomes active. You must determine the system behavior for recovering from a configuration error such as downloading a different BIT file if the error condition is detected.

When you download partial BIT files, you cannot use this methodology for error detection and recovery. The FPGA is by definition already in user mode when the partial BIT file is loaded. Because the configuration circuitry supports error detection only after a BIT file has been loaded, a corrupt partial BIT file can become active, potentially damaging the FPGA if left operating for an extended period of time.

If a CRC error is detected during a partial reconfiguration, it asserts the INIT_B pin of the FPGA (INIT_B goes Low to indicate a CRC error). In UltraScale devices, this behavior is echoed on the PRERROR output pin of the ICAP. It is important to note that if a system monitors INIT_B for CRC errors during the initial configuration, a CRC error during a partial reconfiguration might trigger the same response. To detect the presence of a CRC error from within the FPGA, the CRC status can be monitored through the ICAP block. The Status Register (STAT) indicates that the partial BIT file has a CRC error, by asserting the CRC_ERROR flag (bit 0).

There are two types of partial BIT file errors to consider: data errors and address errors (the partial BIT file is essentially address and data information). Given that static routes are free to pass through reconfigurable regions, both types of errors can corrupt the static design, although the likelihood is very small. The only method for completely safe recovery is to download a new full BIT file to ensure the state of the static logic, which requires the entire FPGA to be reset.

Many systems do not need a complex recovery mechanism because resetting the entire FPGA is not critical, or the partial BIT file is stored locally. In that case, the chance of BIT file corruption is not appreciable. Systems in which the BIT files are at risk of becoming corrupted (such as sending the partial BIT file over a radio link) should use a dedicated silicon feature that avoids the problem.

The configuration engines of 7 series and UltraScale FPGAs and Zynq-7000 AP SoC devices have the ability to perform a frame-by-frame CRC check and do not load a frame into the configuration memory if that CRC check fails. A failure is reported on the INIT_B pin (it is pulled Low) and gives you the opportunity to take the next steps: retry the partial bit file, fall back to a golden partial bit file, etc. The partially loaded reconfiguration region does not have valid programming in it, but the CRC check ensures the remainder of the device (static region and any other reconfigurable modules) stays operational while the system recovers from the error.

To enable this feature for these devices, set the `PerFrameCRC` property prior to running `write_bitstream`. The default is No, and Yes inserts the extra CRC checks. The size of an uncompressed bit file increases four to five percent with this option enabled. No specific design considerations are necessary to select this option, but your partial reconfiguration controller solution should be designed to choose the course of action should the INIT_B pin indicate a failure has occurred.

The syntax for setting the `PerFrameCRC` property is:

```
set_property bitstream.general.perFrameCRC yes [current_design]
```

After a partial bit file has been loaded (with or without the per-frame CRC checks), the overall configuration of the device has changed. If the POST_CRC feature for SEU mitigation is enabled, the SEU mitigation engine automatically recalculates the embedded SEU CRC value after the partial bitstream has been loaded and after you have de-synced the configuration interface. Upon completion of the CRC recalibration, the FRAME_ECCE2 FRAME_VALID output toggles again to indicate that SEU detection has resumed.

Configuration Frames

All user-programmable features inside Xilinx FPGA and AP SoC devices are controlled by volatile memory cells that must be configured at power-up. These memory cells are collectively known as configuration memory. They define the LUT equations, signal routing, IOB voltage standards, and all other aspects of the design.

Xilinx FPGA and AP SoC architectures have configuration memory arranged in frames that are tiled about the device. These frames are the smallest addressable segments of the device configuration memory space, and all operations must therefore act upon whole configuration frames.

Reconfigurable Frames are built upon these configuration frames, and these are the minimum building blocks for performing Partial Reconfiguration.

- Base Regions in 7 series FPGAs are:
  - **CLB**: 50 high by 1 wide
  - **DSP48**: 10 high by 1 wide
  - **Block RAM**: 10 high by 1 wide
- Base Regions in UltraScale™ FPGAs are:
  - **CLB**: 60 high by 1 wide
  - **DSP48**: 24 high by 1 wide
  - **Block RAM**: 12 high by 1 wide
  - **I/O and Clocking**: 52 I/O (one bank), plus related XiPhy, MMCM, and PLL resources
  - **Gigabit Transceivers**: 4 high (one quad, plus related clocking resources)

Configuration Time

The speed of configuration is directly related to the size of the partial BIT file and the bandwidth of the configuration port. The different configuration ports in 7 series architectures have the maximum bandwidths shown in Table 7-2.

Table 7-2: **Maximum Bandwidths for Configuration Ports in 7 Series Architectures**

Configuration Mode	Max Clock Rate	Data Width	Maximum Bandwidth
ICAP	100 MHz	32 bit	3.2 Gb/s
SelectMAP	100 MHz	32 bit	3.2 Gb/s
Serial Mode	100 MHz	1 bit	100 Mb/s
JTAG	66 MHz	1 bit	66 Mb/s

The exact bitstream length is available in the created `.rbt` file by using the `-raw_bitfile` option for `write_bitstream`. Use this number along with the bandwidth to calculate the total configuration time. Here is an example of the header in a raw bit file:

```
Xilinx ASCII Bitstream
Created by Bitstream 2015.1
Design name: led_shift_count;UserID=0XFFFFFFFF
Architecture:kintex7
Part:        7k325tffg900
Date:        Mon Mar 16 16:42:05 2015
Bits:        1211072
11111111111111111111111111111111
```

Configuration Debugging

The ICAP interface can be use used to monitor the configuration process, even if other configuration means are used (JTAG or Slave SelectMAP). In fact, the status of the configuration is automatically pushed out to the “O” port of the ICAP without having to issue a read.

In addition to the techniques described below, the UltraScale architecture introduces two new dedicated ports on the ICAP to aid in Partial Reconfiguration:

- The PRDONE signal echoes the external DONE pin. It drops LOW when reconfiguration starts (at the beginning of the clearing bitstream) and returns HIGH upon successful completion (at the end of the partial bitstream).
- The PRERROR pin echoes the external INIT_B pin. It drops LOW when a CRC error occurs, either with the standard full CRC value at the end of the bit file, or with any per-frame CRC value.

The "O" port of the ICAP block is a 32-bit bus, but only the lowest byte is used. The mapping of the lower byte is as follows:

Table 7-3: **ICAP "O" Port Bits**

ICAP "O" Port Bits	Status Bit	Meaning
O[7]	CFGERR_B	Configuration error (active-Low) 0 = A configuration error has occurred. 1 = No configuration error.
O[6]	DALIGN	Sync word received (active-High) 0 = No sync word received. 1 = Sync word received by interface logic.
O[5]	RIP	Readback in progress (active-High) 0 = No readback in progress. 1 = A readback is in progress.
O[4]	IN_ABORT_B	ABORT in progress (active-Low) 0 = Abort is in progress. 1 = No abort in progress.
O[3:0]	1	Reserved

The most significant nibble of this byte reports the status. These Status bits indicate whether the Sync word been received and whether a configuration error has occurred. The following table displays the values for these conditions.

Table 7-4: **ICAP Sync Bits**

O[7:0]	Sync Word?	CFGERR?
9F	No Sync	No CFGERR
DF	Sync	No CFGERR
5F	Sync	CFGERR
1F	No Sync	CFGERR

Figure 7-5, page 104 shows a completed full configuration, followed by a partial reconfiguration with a CRC error, and finally a successful partial reconfiguration. Using the table above, and the description below, you can see how the "O" port of the ICAP can be used to monitor the configuration process. If a CRC error occurs, these signals can be used by a configuration state machine to recover from the error. These signals can also be used by Vivado Logic Analyzer to capture a configuration failure for debug purposes. With this information Vivado Logic Analyzer can also be used to capture the various points of a partial reconfiguration.

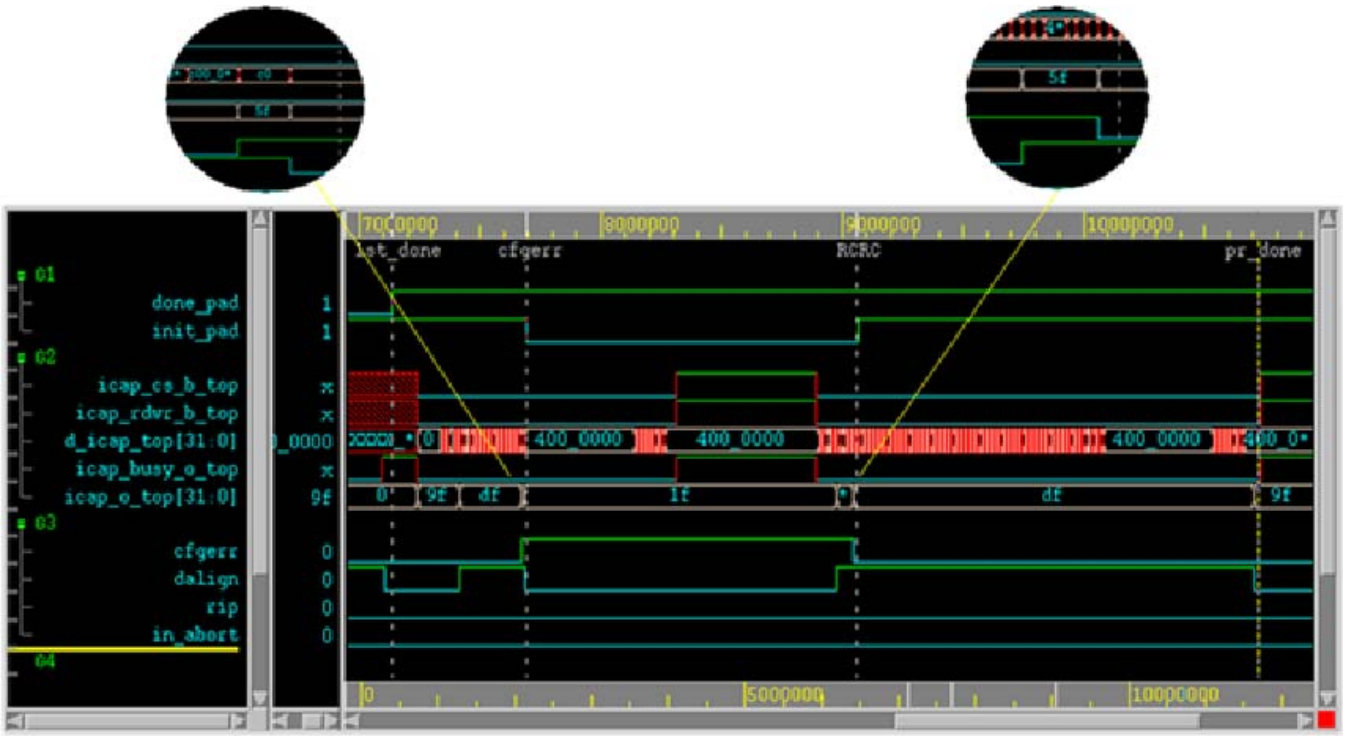

Figure 7-5: **Vivado Logic Analyzer Display for Partial Reconfiguration**

The markers in the Vivado Logic Analyzer display indicate the following:

- **`1st_done`**

  This marker indicates the completion of the initial full bitstream configuration. The DONE pin (`done_pad` in this waveform) goes HIGH.

- **`cfgerr`**

  This marker indicates a CRC error is detected while loading partial bitstream. The status can be observed through O[31:0] (`icap_o_top[31:0]` in the waveform).

  - `Icap_o_top[31:0]` starts at 0x9F
  - After seen SYNC word, `Icap_o_top[31:0]` change to 0xDF
  - After detect CRC error, `Icap_o_top[31:0]` change to 0x5F for one cycle, and then switches to 0x1F
  - INIT_B pin is pulled Low (`init_pad` in the waveform)

- **`RCRC`**

  This marker indicates when the partial bitstream is loaded again. The RCRC command resets the `cfgerr` status, and removes the pull-down on the INIT_B pin (`init_pad` in this waveform).

- `Icap_o_top[31:0]` change from 0x1F to 0x5F when the SYNC word is seen
- `Icap_o_top[31:0]` change from 0x5F to 0xDF when RCRC command is received

- **`pr_done`**

  This marker indicates a successful Partial Reconfiguration.

  - `Icap_o_top[31:0]` change from 0xDF to 0x9F when the DESYNC command is received and no configuration error is detected.

Known Issues and Limitations

Known Issues

This is a list of issues that might be encountered when using Partial Reconfiguration in the current Vivado® Design Suite release. If you encounter any of these issues, or discover any others, please inform Xilinx® and send an example design that shows the issue. These test cases are very helpful for our efforts to improve the overall solution.

- Report to Xilinx all cases of fatal or internal errors, incomplete routing (partial antennas), or other rule violations that prevent place and route, `pr_verify`, and `write_bitstream` from succeeding. Including a design showing the failure is critical for proper analysis and implementation of fixes.
- Reuse of implemented Reconfigurable Modules is not 100% preserved. In a future release, a checkpoint representing an implemented Reconfigurable Module could be saved from one configuration and then reused in another configuration. However, in the current release, the interface nets between the partition pins and the internal logic are not captured, so these signals must be rerouted.
  - This can be done by running `route_design` after loading in a routed RM checkpoint. This process has not been extensively tested and is not recommended.
- If the initial configuration of a 7 series SSI device (7V2000T, 7VX1140T) is done through an SPI interface, partial bitstreams cannot be delivered to the master (or any) ICAP; they must be delivered to an external port, such as JTAG. If the initial configuration is done through any other configuration port, the master ICAP can be used as the delivery port for partial bitstreams.
  - Contact Xilinx Support for a workaround.
- Do not drive multiple outputs of a single Reconfigurable Module with the same source. Each output of an RM must have a unique driver.
- Engineering Silicon (ES) for UltraScale™ devices do not officially support Partial Reconfiguration. To investigate the capabilities of PR on ES devices, please contact Xilinx Support for advice.
- Bottom up synthesis on a non-IP level of hierarchy in IPI is not currently supported.

Blanking Bitstreams Recommended for 7 series and Zynq-7000 Family Partial Reconfiguration Designs

This issue affects designs in all versions of Vivado Design Suite and ISE using Xilinx 7 series and Zynq-7000 devices. Below is a description of the issue and current solution. A future version of the Vivado software will automate the solution without requiring user action.

Description

There is a small probability that during reconfiguration, static signals passing through reconfigurable regions may experience a brief glitch that could disrupt the operation of the static design. Multiple factors must exist for this to occur and the expected failure rate is considered extremely low; these factors include routing resources used, configuration frame ordering, order of the delivery of partial bitstreams, and the current value of the static signal.

Given that the glitch condition is partially based on bitstream construction, the behavior is repeatable – a specific transition from one reconfigurable module to another reconfigurable module (but not back) may exhibit the condition. If hardware testing has not shown any disruption in static design behavior, it is unlikely to be seen in a deployed system. Nevertheless, to ensure the integrity of the static design during reconfiguration, inserting a blanking bitstream is recommended.

Solution

Users can create and deliver black box or "blanking" partial bitstreams to effectively remove all routes in a Reconfigurable Partition (RP) prior to the loading of the next Reconfigurable Module (RM). This blanking bitstream will contain only static routes for the defined region, thus ensuring no glitch behavior can occur. Unlike clearing bitstreams for UltraScale devices, delivery of blanking bitstreams is not order-dependent and only one is required per RP. See Blanking Bitstreams, page 92 for more information.

Creation of the blanking bitstream is simple:

- Vivado: From a routed full design checkpoint, call `update_design -black_box` for each Reconfigurable Partition, a required step in the PR flow, and save the resulting checkpoint. `write_bitstream` creates a blanking bitstream for each RP, and the `-cell` option can be used to target specific RPs.
- ISE: Create an explicit design configuration with black boxes, importing the static results from the primary design configuration. `bitgen` creates a blanking bitstream for each RP. See *Partial Reconfiguration User Guide* (UG702) [Ref 10] for more information.

Bitstream compression can be used to minimize blanking bitstream size. Blanking bitstreams should be considered the same as standard partial bitstreams when it comes to behavior (e.g. DONE) and delivery with one note: Decoupling logic should be enabled prior to loading the blanking bitstream and held through delivery of the subsequent partial bitstream, as the module outputs are undriven for the blanking module. The Partial Reconfiguration Controller (PRC) IP can be used to manage standard and blanking partial bitstreams, however, the user must define the sequence of events outside the PRC. Alternatively, contact Xilinx Support for a more automated approach using the PRC.

A future version of Vivado software will automate this blanking process without requiring user action.

Known Limitations

Certain features are not yet developed or supported in the current release. Some of these features might be added in upcoming releases. These include:

- When selecting Pblock ranges to define the size and shape of the Reconfigurable Partition, do not use the CLOCKREGION resource type for 7 series or Zynq designs. Pblock ranges must only include types SLICE, RAMB18, RAMB36, and DSP48 resource types.
- Project support. Compiling configurations using projects and project commands (`create_run`, `launch_runs`, etc.) is not yet supported. Managing PR projects in the Vivado IDE is likewise not yet supported.

  Checkpoints can be opened in the IDE and many analysis features can be used, but the Design Runs features cannot be used.
- Do not use Vivado Debug core insertion features within Reconfigurable Partitions. This flow inserts the debug hub, which includes BSCAN primitive, which is not permitted inside reconfigurable bitstreams.
- Do not use Partial Reconfiguration with Tandem Configuration capabilities within Xilinx PCIe® IP. For more information regarding Tandem Configuration with Field Updates, see the *UltraScale Architecture Gen3 Integrated Block for PCI Express* (PG156) [Ref 28].
- UpdateMEM does not support partial bitstreams. To associate memory files with PR designs, the ELF Association flow must be applied. See this link in the *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator* (UG994) [Ref 30] for details.
- The Soft Error Mitigation (SEM) IP core is supported in conjunction with PR in monolithic devices. For more information on using the SEM IP in PR designs, see *Demonstration of Soft Error Mitigation IP and Partial Reconfiguration Capability on Monolithic Devices* (XAPP1261) [Ref 4]. The SEM IP core is not supported when using Partial Reconfiguration on SSI devices.

- Two use cases regarding encryption will not be supported when using new features within UltraScale devices:

  a. If RSA authentication is selected for the initial configuration, then encrypted partial reconfiguration is not supported. RSA authentication is not supported for partial bitstreams.

  b. If the initial configuration bitstream uses an obfuscated AES-256 key stored in either the eFUSE or BBRAM, then any encrypted partial bitstreams must use the same obfuscated key. Encrypted PR bitstreams using a different key than the initial bitstream is not supported.

  In either of these two cases, an unencrypted partial bitstream may be delivered to the ICAP to partially reconfigure the device.

Appendix A

Additional Resources and Legal Notices

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see Xilinx Support.

Solution Centers

See the Xilinx Solution Centers for support on devices, software tools, and intellectual property at all stages of the design cycle. Topics include design assistance, advisories, and troubleshooting tips.

References

1. *Vivado Design Suite Tutorial: Partial Reconfiguration* (UG947)
2. *Partial Reconfiguration Controller IP* (PG193)
3. *Partial Reconfiguration Decoupler IP* (PG227)
4. *Demonstration of Soft Error Mitigation IP and Partial Reconfiguration Capability on Monolothic Devices* (XAPP1261)
5. *Bitstream Loading across the PCI Express Link in UltraScale Devices for Tandem PCIe and Partial Reconfiguration* (AR# 64761)
6. *Partial Reconfiguration of a Hardware Accelerator with Vivado Design Suite for Zynq-7000 AP SoC Processor* (XAPP1231)
7. *7 Series FPGAs Configuration User Guide* (UG470)
8. *UltraScale Architecture Configuration User Guide* (UG570)
9. *Zynq-7000 All Programmable SoC Technical Reference Manual* (UG585)
10. *Partial Reconfiguration User Guide* (UG702) - For ISE Design Tools

11. *Hierarchical Design Methodology Guide* (UG748) - For ISE Design Tools
12. *UltraFast Design Methodology Guide for the Vivado Design Suite* (UG949)
13. *7 Series FPGAs Integrated Block for PCI Express Product Guide* (PG054)
14. *Virtex-7 FPGA Gen3 Integrated Block for PCI Express Product Guide* (PG023)
15. *LogiCORE IP UltraScale FPGAs Gen3 Integrated Block for PCI Express Product Guide* (PG156)
16. *Vivado Design Suite Tcl Command Reference Guide* (UG835)
17. *Vivado Design Suite User Guide: Synthesis* (UG901)
18. *Vivado Design Suite User Guide: Using Constraints* (UG903)
19. *Vivado Design Suite User Guide: Design Analysis and Closure Techniques* (UG906)
20. *7 Series FPGAs GTX/GTH Transceivers User Guide* (UG476)
21. *7 Series FPGAs GTP Transceivers User Guide* (UG482)
22. *MMCM and PLL Dynamic Reconfiguration (7 Series)* (XAPP888)
23. *UltraScale Architecture Clocking Resources User Guide* (UG572)
24. *UltraScale Architecture GTH Transceivers User Guide* (UG576)
25. *UltraScale Architecture GTY Transceivers User Guide* (UG578)
26. *PRC/EPRC: Data Integrity and Security Controller for Partial Reconfiguration* (XAPP887)
27. *Vivado Design Suite User Guide: Programming and Debugging* (UG908)
28. *UltraScale Architecture Gen3 Integrated Block for PCI Express* (PG156)
29. *UltraScale Architecture Configurable Logic Block User Guide* (UG574)
30. *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator* (UG994)
31. *Design Advisory for techniques on properly synchronizing flip-flops and SRLs* (AR# 44174)
32. Vivado Design Suite Documentation

Training Resources

Xilinx provides a variety of training courses and QuickTake videos to help you learn more about the concepts presented in this document. Use these links to explore related training resources:

1. Vivado Design Suite QuickTake Video Tutorials
2. Vivado Design Suite QuickTake Video: Partial Reconfiguration in Vivado
3. Partial Reconfiguration Flow on Zynq using Vivado
4. Xilinx Partial Reconfiguration Tools and Techniques

Please Read: Important Legal Notices