Partial Reconfiguration

Updated for Intel® Quartus® Prime Design Suite: 20.3
# Contents

1. Creating a Partial Reconfiguration Design .................................................. 4
   1.1. Partial Reconfiguration Terminology ..................................................... 5
   1.2. Partial Reconfiguration Process Sequence ........................................... 6
   1.3. Internal Host Partial Reconfiguration .................................................... 7
   1.4. External Host Partial Reconfiguration .................................................... 9
   1.5. Partial Reconfiguration Design Flow .................................................... 10
      1.5.1. Step 1: Identify Partial Reconfiguration Resources ......................... 11
      1.5.2. Step 2: Create Design Partitions .................................................... 11
      1.5.3. Step 3: Floorplan the Design ......................................................... 13
      1.5.4. Step 4: Add the Partial Reconfiguration Controller Intel FPGA IP ............ 16
      1.5.5. Step 5: Define Personas ................................................................. 18
      1.5.6. Step 6: Create Revisions for Personas ............................................ 18
      1.5.7. Step 7: Compile the Base Revision and Export the Static Region .......... 20
      1.5.8. Step 8: Setup PR Implementation Revisions ..................................... 22
      1.5.9. Step 9: Program the FPGA Device ................................................ 23
   1.6. Partial Reconfiguration Design Considerations ....................................... 29
      1.6.1. Partial Reconfiguration Design Guidelines ....................................... 31
      1.6.2. PR Design Timing Closure Best Practices ......................................... 32
      1.6.3. PR File Management ..................................................................... 34
      1.6.4. Evaluating PR Region Initial Conditions ........................................... 38
      1.6.5. Creating Wrapper Logic for PR Regions .......................................... 38
      1.6.6. Creating Freeze Logic for PR Regions ............................................ 39
      1.6.7. Resetting the PR Region Registers .................................................. 40
      1.6.8. Promoting Global Signals in a PR Region ......................................... 41
      1.6.9. Planning Clocks and other Global Routing ....................................... 42
      1.6.10. Implementing Clock Enable for On-Chip Memories with Initialized Contents ... 43
   1.7. Hierarchical Partial Reconfiguration ....................................................... 46
      1.7.1. Using Parent QDB Files from Different Compiles ............................. 46
   1.8. Partial Reconfiguration Design Timing Analysis ..................................... 47
      1.8.1. Running Timing Analysis on Aggregate Revisions ............................. 48
   1.9. Partial Reconfiguration Design Simulation ........................................... 49
      1.9.1. Partial Reconfiguration Simulation Flow ......................................... 49
   1.10. Partial Reconfiguration Design Debugging ......................................... 54
      1.10.1. Debugging PR Designs with the Signal Tap Logic Analyzer ............... 54
   1.11. PR Bitstream Security Verification (Intel Stratix 10 Designs) .................... 56
      1.11.1. PR Bitstream Security Use Case (Intel Stratix 10 Designs) .................. 56
      1.11.2. Using PR Bitstream Security Verification (Intel Stratix 10 Designs) ....... 57
   1.12. PR Bitstream Compression and Encryption (Intel Arria 10 and Intel Cyclone 10 GX Designs) ................................................................. 58
      1.12.1. Generating an Encrypted PR Bitstream (Intel Arria 10 or Intel Cyclone 10 GX Designs) ................................................................. 59
      1.12.2. Clock-to-Data Ratio for Bitstream Encryption and Compression (Intel Arria 10 or Intel Cyclone 10 GX Designs) ........................................... 60
      1.12.3. Data Compression Comparison ..................................................... 61
   1.13. Avoiding PR Programming Errors ....................................................... 61
   1.14. Exporting a Version-Compatible Compilation Database for PR Designs .... 63
1. Creating a Partial Reconfiguration Design

Partial reconfiguration (PR) allows you to reconfigure a portion of the FPGA dynamically, while the remaining FPGA design continues to function. You can define multiple personas for a particular region in your design, without impacting operation in areas outside this region. This methodology is effective in systems with multiple functions that time-share the same FPGA device resources. PR enables the implementation of more complex FPGA systems.

The Intel® Quartus® Prime Pro Edition software supports the PR feature for the Intel Stratix® 10, Intel Agilex™(1), Intel Arria® 10, and Intel Cyclone® 10 GX device families.

Figure 1. Partial Reconfiguration Design

PR provides the following advancements over a flat design:

- Allows run-time design reconfiguration
- Increases scalability of the design through time-multiplexing
- Lowers cost and power consumption through efficient use of board space
- Supports dynamic time-multiplexing functions in the design
- Improves initial programming time through smaller bitstreams
- Reduces system down-time through line upgrades
- Enables easy system update by allowing remote hardware change
- A simplified compilation flow for partial reconfiguration

(1) The Intel Quartus Prime Pro Edition software supports compilation of PR designs for Intel Agilex devices, but does not yet support Intel Agilex PR bitstream generation.

ISO 9001:2015 Registered
Hierarchical Partial Reconfiguration

Intel Quartus Prime Pro Edition software also supports hierarchical partial reconfiguration (HPR), with multiple parent and child design partitions, or multiple levels of partitions in a design. In HPR designs, a static region instantiates a parent PR region, and a parent PR region instantiates a child PR region. The same PR region reprogramming is possible for the child and parent partitions. Refer to Hierarchical Partial Reconfiguration on page 46.

Static Update Partial Reconfiguration

Static update partial reconfiguration (SUPR) allows you to define and modify a specialized static region, without requiring recompilation of all personas. This technique is useful for a portion of a design that you may possibly want to change for risk mitigation, but that never requires runtime reconfiguration. In PR without an SUPR partition, you must recompile all personas for any change to the static region. Refer to the Partial Reconfiguration Tutorials for detailed SUPR instructions.

Partial Reconfiguration Design Simulation

The Intel Quartus Prime Pro Edition software supports simulation of PR persona transitions through use of simulation multiplexers. You use the simulation multiplexers to change which persona drives logic inside the PR region during simulation. This simulation allows you to observe the resulting change and the intermediate effect in a reconfigurable partition. Refer to Partial Reconfiguration Design Simulation on page 49 for details.

Related Information

Partial Reconfiguration Tutorials

1.1. Partial Reconfiguration Terminology

This document refers to the following terms to explain partial reconfiguration:

Table 1. Partial Reconfiguration Terminology

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Floorplan</td>
<td>The layout of physical resources on the device. Creating a design floorplan, or floorplanning, is the process of mapping the logical design hierarchy to physical regions in the device.</td>
</tr>
<tr>
<td>Hierarchical Partial Reconfiguration</td>
<td>Partial reconfiguration that includes multiple parent and child design partitions, or nesting of partitions in the same design.</td>
</tr>
<tr>
<td>PR control block</td>
<td>A dedicated block in Intel Arria 10 and Intel Cyclone 10 GX FPGAs. The PR control block processes the PR requests, handshake protocols, and verifies the cyclic redundancy check (CRC).</td>
</tr>
<tr>
<td>PR host</td>
<td>The system for coordinating PR. The PR host communicates with the PR control block (Intel Arria 10 and Intel Cyclone 10 GX designs) or Secure Device Manager (Intel Stratix 10 and Intel Agilex designs). Implement the PR host within the FPGA (internal PR host) or in a chip or microprocessor.</td>
</tr>
<tr>
<td>PR partition</td>
<td>Design partition that you designate as Reconfigurable. A PR project can contain one or more PR partitions.</td>
</tr>
<tr>
<td>PR Solutions Intel FPGA IP</td>
<td>Suite of Intel FPGA IP that simplify implementation of PR handshaking and freeze logic, as Partial Reconfiguration Solutions IP User Guide on page 69 describes.</td>
</tr>
</tbody>
</table>

continued...
### Term | Description
--- | ---
**PR region** | A physical portion of FPGA device that you designate for partial reconfiguration. You define a PR region in the base configuration design. A device can contain more than one PR region. A PR region can be core-only, such as LAB, RAM, or DSP. The PR region bitstream configures this region.

**PR persona** | A specific PR partition implementation in a PR region. A PR region can contain multiple personas. Static regions contain only one persona.

**Revision** | A collection of settings and constraints for one version of your project. An Intel Quartus Prime Settings File (.qsf) preserves each revision of your project. Your Intel Quartus Prime project can contain several revisions. Revisions allow you to organize several versions of your design within a single project.

**Secure Device Manager (SDM)** | A triple-redundant processor-based block in Intel Agilex and Intel Stratix 10 devices that performs authentication, decryption, and decompression on the configuration data the block receives, before sending the data over to the configurable nodes through the configuration network.

**Snapshot** | The output of a Compiler stage. You can export the synthesis or final compilation results snapshot.

**Static region** | All areas not occupied by PR regions in your project. You associate the static region with the top-level partition of the design. The static region contains both the core and periphery locations of the device. The static region bitstream configures this region.

**Static update partial reconfiguration** | A static region that allows change, without requiring the recompilation of all personas. This technique is useful for a portion of a design that you may possibly want to change for risk mitigation, but that never requires runtime reconfiguration.

### 1.2. Partial Reconfiguration Process Sequence

Your partial reconfiguration design must initiate the PR operation and deliver the configuration file to the PR control block (Intel Arria 10 and Intel Cyclone 10 GX designs) or SDM (Intel Agilex and Intel Stratix 10 designs). Before partial reconfiguration, you must ensure that the FPGA device is in user mode, and in a functional state. The following steps describe the partial reconfiguration sequence:

1. **Send the stop_req signal to the PR region from the sequential PR control logic to prepare for the PR operation.** Upon receiving this signal, the PR regions complete any pending transactions and stop accepting new transactions.

2. **Wait for the stop_ack signal to indicate that the PR region is ready for partial reconfiguration.**

3. **Use PR control logic to freeze all necessary outputs of the PR regions.** Additionally, drive the clock enable for any initialized RAMs to disabled state.

4. **Send the PR bitstream to the PR control block (Intel Arria 10 and Intel Cyclone 10 GX designs) or SDM (Intel Stratix 10 and Intel Agilex designs) to initiate the PR process for the PR region.** When using any of the Partial Reconfiguration Controller Intel FPGA IP, the Avalon® memory-mapped or Avalon streaming interface on the IP core provides this functionality. When directly instantiating the PR control block for Intel Arria 10 designs, refer to **PR Control Block Signal Timing Diagrams** on page 90.

5. **On successful completion of the PR operation, reset the PR region.**
6. Signal the start of PR operation by asserting the `start_req` signal, and deasserting the `freeze` signal.

7. Wait for the `start_ack` signal to indicate that the PR region is ready for operation.

8. Resume operation of the FPGA with the newly reconfigured PR region.

![PR Process Sequence Timing Diagram](image)

**Figure 2. PR Process Sequence Timing Diagram**

6. Signal the start of PR operation by asserting the `start_req` signal, and deasserting the `freeze` signal.

7. Wait for the `start_ack` signal to indicate that the PR region is ready for operation.

8. Resume operation of the FPGA with the newly reconfigured PR region.

**1.3. Internal Host Partial Reconfiguration**

In internal host control, an internal controller, a Nios® II processor, or an interface such as PCI Express® (PCIe®) or Ethernet, communicates directly with the Intel Arria 10 or Intel Cyclone 10 GX PR control block, or with the SDM in Intel Stratix 10 and Intel Agilex devices.

To transfer the PR bitstream into the PR control block or SDM, you use the Avalon memory-mapped interface on the Partial Reconfiguration Controller IP core. When the device enters user mode, you initiate partial reconfiguration through the FPGA core fabric using the PR internal host.

**Note:** If you create your own control logic for the PR host, the logic must meet the PR interface requirements.
When performing partial reconfiguration with an internal host, use the dedicated PR pins (PR_REQUEST, PR_READY, PR_DONE, and PR_ERROR) as regular I/Os. Implement your static region logic to retrieve the PR programming bitstreams from an external memory, for processing by the internal host.

Send the programming bitstreams for partial reconfiguration through the PCI Express link. Then, you process the bitstreams with your PR control logic and send the bitstreams to the PR IP core for programming. nCONFIG moves the device out of the user mode into the device configuration mode.\(^{(2)}\)

\(^{(2)}\) nCONFIG can lock the device and force a power-cycle. PR programming may corrupt the static logic, due to improper use, causing disconnection of the core clock input to configuration block and unresponsive configuration. You must reset the PR IP before toggling nCONFIG.
1.4. External Host Partial Reconfiguration

In external host control, an external FPGA or CPU controls the PR configuration using external dedicated PR pins on the target device. When using an external host, you must implement the control logic for transmission of the bitstream to the hard FPGA programming pins.

Figure 5. PR System Using an External Host (Intel Arria 10 Example)

Related Information
- Configuring an External Host for Intel Arria 10 or Intel Cyclone 10 GX Designs on page 93
- Configuring an External Host for Intel Stratix 10 or Intel Agilex Designs on page 97
1.5. Partial Reconfiguration Design Flow

The PR design flow requires initial planning. This planning involves setting up one or more design partitions, and then determining the placement assignments in the floorplan. Well-planned PR partitions improve design area utilization and performance. The Intel Quartus Prime software also allows you to create nested PR regions as part of an HPR flow.

Figure 6. Partial Reconfiguration Design Flow

The PR design flow uses the project revisions feature in the Intel Quartus Prime software. Your initial design is the base revision, where you define the static region boundaries and reconfigurable regions on the FPGA. From the base revision, you
create multiple revisions. These revisions contain the different implementations for the PR regions. However, all PR implementation revisions use the same top-level placement and routing results from the base revision.

The PR design flow includes the following steps:

- Step 1: Identify Partial Reconfiguration Resources on page 11
- Step 2: Create Design Partitions on page 11
- Step 3: Floorplan the Design on page 13
- Step 4: Add the Partial Reconfiguration Controller Intel FPGA IP on page 16
- Step 5: Define Personas on page 18
- Step 6: Create Revisions for Personas on page 18
- Step 7: Compile the Base Revision and Export the Static Region on page 20
- Step 8: Setup PR Implementation Revisions on page 22
- Step 9: Program the FPGA Device on page 23

1.5.1. Step 1: Identify Partial Reconfiguration Resources

When designing for partial reconfiguration, you must first determine the logical hierarchy boundaries that you can define as reconfigurable partitions. Reconfigurable partitions must contain only core resources, such as LABs, embedded memory blocks (M20Ks and MLABs), and DSP blocks in the FPGA.

All periphery resources, such as transceivers, external memory interfaces, GPIOs, I/O receivers, and hard processor system (HPS), must be in the static region. Partial reconfiguration of global network buffers for clocks and resets is not possible.

<table>
<thead>
<tr>
<th>Hardware Resource Block</th>
<th>Reconfiguration Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logic Block</td>
<td>Partial reconfiguration</td>
</tr>
<tr>
<td>Digital Signal Processing</td>
<td>Partial reconfiguration</td>
</tr>
<tr>
<td>Memory Block</td>
<td>Partial reconfiguration</td>
</tr>
<tr>
<td>Core Routing</td>
<td>Partial reconfiguration</td>
</tr>
<tr>
<td>Transceivers/PLL</td>
<td>Dynamic reconfiguration</td>
</tr>
<tr>
<td>I/O Blocks</td>
<td>Not supported</td>
</tr>
<tr>
<td>Clock Control Blocks</td>
<td>Not supported</td>
</tr>
</tbody>
</table>

After identifying the resources for PR, set up the design hierarchy and source code to support this partitioning. Refer to Partial Reconfiguration Design Considerations on page 29.

1.5.2. Step 2: Create Design Partitions

Create design partitions for each PR region that you want to partially reconfigure. Create any number of independent partitions or PR regions in your design. Create design partitions for partial reconfiguration from the Project Navigator, or the Design Partitions Window.
A design partition is only a logical partitioning of the design, and does not specify a physical area on the device. You associate a partition with a specific area of the FPGA using Logic Lock Region assignments. To avoid partitions obstructing design optimization, group the logic together within the same partition. If your design includes a hierarchical PR flow with parent and child partitions, you can define multiple parent or child partitions in your design, and multiple levels of PR partitions.

When you create a **Reconfigurable** partition, the Compiler preserves post-synthesis results for the partition and reuses the post-synthesis netlist, if you make no partition changes requiring re-synthesis. Otherwise, the Compiler resynthesizes the partition from source files. The Compiler adds wire LUTs for each interface of a **Reconfigurable** partition, and performs checks for PR compatibility. You must run elaboration before creating partitions.

**Figure 7. Creating a Design Partition**

Follow these steps to create design partitions:

1. Click **Processing ➤ Start ➤ Start Analysis & Elaboration**.
2. In the Project Navigator, right-click an instance in the **Hierarchy** tab, click **Design Partition ➤ Set as Design Partition**. A design partition icon appears next to each partition you create.
3. To view and edit all design partitions in the project, click **Assignments ➤ Design Partitions Window**.
4. Specify **Reconfigurable** as the partition **Type** for each PR partition. The **Reconfigurable** type preserves synthesis results, while allowing refit of the partition in the PR flow.
1.5.3. Step 3: Floorplan the Design

Use Logic Lock floorplan constraints in your PR design to physically partition the device. Each PR partition in your design must have a corresponding, exclusive physical partition. You create Logic Lock regions to define the physical partition for your PR region. This partitioning ensures that the resources available to the PR region are the same for any persona that you implement.
Your PR region must include only core logic, such as LABs, RAMs, ROMs, and DSPs in a PR region. Intel Agilex and Intel Stratix 10 designs can also include Hyper-Registers in the PR partition. Instantiate all periphery design elements, such as transceivers, external memory interfaces, and clock networks in the static region of the design. The Logic Lock regions you create can cross periphery locations, such as the I/O columns and the HPS, because the constraint is core-only.

There are two region types:

- **Place regions**—use these regions to constrain logic to a specific area of the device. The Fitter places the logic in the region you specify. The Fitter can also place other logic in the region unless you designate the region as **Reserved**.
- **Route regions**—use these regions to constrain routing to a specific area. The routing region must fully enclose the placement region. Additionally, the routing regions for the PR regions cannot overlap.

**Figure 10. Floorplanning your PR Design**

Follow these guidelines when floorplanning your PR design:

- Complete the periphery and clock floorplan before core floorplanning. You can use Interface Planner (`Tools ➤ Interface Planner`) to create periphery floorplan assignments for your design.
- Define a routing region that is at least 1 unit larger than the placement region in all directions.
- Do not overlap the routing regions of multiple PR regions.
- Select the PR region row-wise for least bitstream overhead. In Intel Arria 10 and Intel Cyclone 10 GX devices, short, wider regions generate smaller bitstreams than tall, narrower regions. Intel Agilex and Intel Stratix 10 configuration occurs on sectors. For the least bitstream overhead, ensure that you align the PR region to sectors. Refer to "Analyzing and Optimizing the Design Floorplan," in *Intel Quartus Prime Pro Edition User Guide: Design Optimization*. 
• For Intel Arria 10 and Intel Cyclone 10 GX devices, the height of your floorplan affects the reconfiguration time. A floorplan larger in the \( Y \) direction takes longer to reconfigure. This condition does not apply to Intel Agilex or Intel Stratix 10 devices because they configure according to sectors.

• Define sub Logic Lock regions within PR regions to improve timing closure.

• If your design includes HPR parent and child partitions, the placement region of the parent region must fully enclose the routing and placement region of its child region. Also, the parent wire LUTs must be in an area outside the child PR region. This requirement is because the child PR region is exclusive to all other logic, which includes the parent and the static region.

Related Information
• Intel Quartus Prime Pro Edition User Guide: Design Optimization
• Intel Quartus Prime Pro Edition User Guide: Design Constraints

1.5.3.1. Applying Floorplan Constraints Incrementally

PR implementation requires additional constraints that identify the reconfigurable partitions of the design and device. These constraints significantly impact the Compiler’s timing closure ability. You can avoid and more easily correct timing closure issues by incrementally implementing each constraint, running the Compiler, then verifying timing closure.

**Note:** PR designs require a more constrained floorplan, compared to a flat design. The overall density and performance of a PR design may be lower than an equivalent flat design.

The following steps describe incrementally developing the requirements for your PR design:

1. Implement the base revision using the most complex persona for each PR partition. This initial implementation must include the complete design with all periphery constraints, and top-level .sdc timing constraints. Do not include any Logic Lock region constraints for the PR regions with this implementation.

2. Create partitions by setting the region **Type** option to **Default** in the Design Partitions Window, for all the PR partitions.

3. Register the boundaries of each partition to ensure adequate timing margin.

4. Verify successful timing closure using the Timing Analyzer.

5. Ensure that all the desired signals are driven on global networks. Disable the **Auto Global Clock** option in the Fitter ([Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Fitter)]), to avoid promoting non-global signals.

6. Create Logic Lock core-only placement regions for each of the partitions.

7. Recompile the base revision with the Logic Lock constraints, and then verify timing closure.

8. Enable the **Reserved** option for each Logic Lock region to ensure the exclusive placement of the PR partitions within the placement regions. Enabling the **Reserved** option avoids placing the static region logic in the placement region of the PR partition.
9. Recompile the base revision with the **Reserved** constraint, and then verify timing closure.

10. In the Design Partitions Window, specify the **Type** for each of the PR partitions as **Reconfigurable**. This assignment ensures that the Compiler adds wire LUTs for each interface of the PR partition, and performs additional compilation checks for partial reconfiguration.

11. Recompile the base revision with the **Reconfigurable** constraint, and then verify timing closure. You can now export the top-level partition for reuse in the PR implementation compilation of the different personas.

### 1.5.4. Step 4: Add the Partial Reconfiguration Controller Intel FPGA IP

Depending on the target device family, you can add the Partial Reconfiguration Controller Intel Arria 10/Cyclone 10 FPGA IP or the Partial Reconfiguration Controller Intel FPGA IP to your design to send the partial reconfiguration bitstream to the PR control block or SDM in an internal host configuration.

#### 1.5.4.1. Adding the Partial Reconfiguration Controller Intel FPGA IP

You can customize and instantiate the Partial Reconfiguration Controller Intel FPGA IP from the IP Catalog (**Tools ➤ IP Catalog**).

The Partial Reconfiguration Controller Intel FPGA IP interfaces with the Secure Device Manager (SDM) to manage the bitstream source. The SDM performs authentication and decompression on the configuration data. You can use this IP core in an Intel Agilex or Intel Stratix 10 design when performing partial reconfiguration with an internal PR host, Nios II processor, PCI Express, or Ethernet interface.

**Figure 11.** Partial Reconfiguration Controller (Avalon Streaming Interface)

(3)

The Intel Quartus Prime software supports PR over the core interface using the PR Controller IP core, or PR over the JTAG device pins. PR over JTAG pins does not require instantiation of the Partial Reconfiguration Controller Intel FPGA IP.

---

(3) Avalon memory-mapped interface variant also available.
1.5.4.2. Adding the Partial Reconfiguration Controller Intel Arria 10/Cyclone 10 FPGA IP

The Partial Reconfiguration Controller Intel Arria 10/Cyclone 10 FPGA IP interfaces with the Intel Arria 10 or Intel Cyclone 10 GX PR control block to manage the bitstream source.

Use this IP core in Intel Arria 10 or Intel Cyclone 10 GX designs when performing partial reconfiguration with an internal PR host, Nios II processor, PCI Express, or Ethernet interface.

During partial reconfiguration, you send a PR bitstream stored outside the FPGA to the PR control block inside the FPGA. This communication enables the control block to update the CRAM bits necessary for reconfiguring the PR region in the FPGA. The PR bitstream contains the instructions (opcodes) and the configuration bits necessary for reconfiguring a specific PR region.

![Partial Reconfiguration Controller Intel Arria 10/Cyclone 10 FPGA IP](image)

Instantiate the IP core from the Intel Quartus Prime IP Catalog (Tools ➤ IP Catalog) to automatically connect the IP to the Intel Arria 10 or Intel Cyclone 10 GX PR control block.

If you create your own custom logic to perform the function of the IP core, manually instantiate the control block to communicate with the FPGA system.

Related Information
- Partial Reconfiguration Controller Intel Arria 10/Cyclone 10 FPGA IP on page 77
- PR Control Block and CRC Block Verilog HDL Manual Instantiation on page 87
- PR Control Block and CRC Block VHDL Manual Instantiation on page 88
1.5.5. Step 5: Define Personas

Your partial reconfiguration design can have multiple PR partitions, each with multiple personas. You define the unique function of each persona in separate Verilog HDL, SystemVerilog HDL, or VHDL design files in the project directory. All the PR personas must use the same set of signals to interact with the static region.

Ensure that the signals interacting with the static region are a super-set of all the signals in all the personas. A PR design requires an identical I/O interface for each persona in the PR region. If all personas for your design do not have identical interfaces, you must also create wrapper logic to interface with the static region.

Note: If using the Intel Quartus Prime Text Editor, disable Add file to current project when saving the files. These persona source files should not be part of the Intel Quartus Prime project or compilations.

1.5.6. Step 6: Create Revisions for Personas

Create a base revision for the design, as well as PR implementation revisions for each of the personas. When you define revisions in the GUI or at the command line, the Intel Quartus Prime software automatically adds these assignments required for PR implementation:

- **Entity Rebinding assignment (ENTITY_REBINDING)**—for each PR partition, the software adds an entity rebinding assignment with a place holder for the entity name. Your design may not require all of the entity rebinding assignments of each PR partition, based on the design and the implementation revision. For example, in HPR designs that use the default persona for the parent partition, you add the .qdb file for PR parent, and then use entity rebinding only for the child.
- **QDB File Partition assignment (QDB_FILE_PARTITION)**—the software adds this assignment for the static region, if you specify a .qdb file name.
- **Revision Type Assignment (REVISION_TYPE)**

To create the PR implementation revisions:

1. Click **Project ➤ Revisions**.
2. To create a new revision, double-click <<new revision>>.
3. Specify a unique **Revision name**.
4. Select an existing revision for the Based on revision option.
5. For the **Revision type**, select **Partial Reconfiguration - Base** for the base revision or **Partial Reconfiguration - Persona Implementation** for an implementation revision.
6. Click **Apply** and **OK**.
The following assignments in the respective revision's .qsf file correspond to specifying the revision type from the Settings dialog box:

**Base Revision Assignment:**

```bash
set_global_assignment -name REVISION_TYPE PR_BASE
```

**Implementation Revision Assignment:**

```bash
set_global_assignment -name REVISION_TYPE PR_IMPL
```

For each PR partition, the Intel Quartus Prime software also adds the entity rebinding assignment to the .qsf:

```bash
set_instance_assignment -name ENTITY_REBINDING <entity_name> -to <hierarchical_path>
```

If you base a new implementation revision on an existing .qdb file, The Intel Quartus Prime software also adds the .qdb file partition assignment, with a placeholder for the file name:

```bash
set_instance_assignment -name QDB_FILE_PARTITION <QDB file name>
```

As an example, to create a new implementation revision that uses a .qdb file from a base revision, use the following command:

```bash
create_revision impl_new -based_on <base_revision> \
-new_rev_type impl -root_partition_qdb_file base_static.qdb
```
• `impl_new`—specifies the name of a new implementation revision.

• `-based_on <based_on_revision>` — specifies the PR base revision that the new `impl` revision is based on. Some global assignments from the `based_on` revision are copied over to the `impl` revision. Placeholder entity rebinding assignments are created in the `impl` revision for each PR partition in the base.

• `-new_rev_type <rev_type>` — only useful `rev_type` is `impl`.

• `root_partition_qdb_file <qdb_file>`— creates a `QDB_FILE_PARTITION` assignment in `impl` revision with the specified `.qdb` file.

![Figure 14. Partial Reconfiguration Compilation Flow](image)

**1.5.7. Step 7: Compile the Base Revision and Export the Static Region**

After defining and floorplanning PR partitions and revisions, you compile the base revision and export the static region. You can export individual design partitions manually, or you can export one or more partitions automatically each time you run the Compiler.

Follow these steps to compile and export the base and static region:

1. To specify the current revision, click **Project ➤ Revisions**, and then set the base revision as current, or select the base revision from the main toolbar drop-down list.

2. For Intel Arria 10 and Intel Cyclone 10 GX designs, you can optionally add the following assignments to the `.qsf` to automatically generate the required PR bitstreams following compilation. This step is not required for Intel Stratix 10 or Intel Agilex designs.

   ```
   set_global_assignment -name GENERATE_PR_RBF_FILE ON
   set_global_assignment -name ON_CHIP_BITSTREAM_DECOMPRESSION OFF
   ```

3. To compile the base revision, click **Processing ➤ Start Compilation**.

4. To export the static region, click **Project ➤ Export Design Partition** and specify options for the partition export:
5. Alternatively, follow these steps to automatically export one or more design partitions after each compilation. You can automatically export any design partition that does not have a preserved parent partition, including the root_partition.

a. To open the Design Partitions Window, click Assignments ➤ Design Partitions Window.

b. To automatically export a partition with synthesis results after any time you run synthesis, specify the .qdb export path and file name for the Post Synthesis Export File option for that partition. If you specify only a file name without a path, the file exports to the project directory after compilation.

c. To automatically export a partition with final snapshot results any time you run the Fitter, specify a .qdb file name for the Post Final Export File option for that partition.

Table 3. Design Partition Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partition name</td>
<td>Select root_partition.</td>
</tr>
<tr>
<td>Partition database file</td>
<td>Specify a descriptive file name.</td>
</tr>
<tr>
<td>Include entity-bound SDC files</td>
<td>Enable to include entity bound .sdc files with the partition export.</td>
</tr>
<tr>
<td>Note:</td>
<td>You must enable this option when exporting the base revision (root partition), so that the implementation compiles inherit the timing constraints defined in entity-bound .sdc files.</td>
</tr>
<tr>
<td>Snapshot</td>
<td>Select final snapshot.</td>
</tr>
</tbody>
</table>
1.5.8. Step 8: Setup PR Implementation Revisions

You must prepare the PR implementation revisions before you can generate the PR bitstream for device programming. This setup includes adding the static region .qdb file as the source file for each implementation revision. In addition, you must specify the corresponding entity of the PR region.

Follow these steps to setup the PR implementation revisions:

1. Set an implementation revision as the **Current Revision**.

2. To specify the .qdb file as the source for root_partition, click **Assignments ➤ Design Partitions Window**. Double-click the **Partition Database File** cell and specify the appropriate .qdb file.

3. For each PR implementation revision, specify the name of the entity that you want to partially reconfigure in the **Entity Re-binding cell**. This entity name comes from the design file for the persona you want to implement in this implementation revision.

4. To compile the design, click **Processing ➤ Start Compilation**.

5. Repeat steps 1 through 4 to setup and compile each implementation revision. Alternatively, use a simple Tcl script to compile all implementation revisions:
1.5.9. Step 9: Program the FPGA Device

The Intel Quartus Prime Assembler generates the PR bitstreams for your design personas. For Intel Arria 10 and Intel Cyclone 10 GX designs, you send the bitstreams to the PR control block. For Intel Stratix 10 and Intel Agilex designs, you send the PR bitstreams to the SDM. You must compile the PR project, including the base revision, and at least one implementation revision, before generating the PR bitstreams.

For Intel Stratix 10 and Intel Agilex designs, the Assembler generates a configuration .rbf automatically at the end of compilation. For Intel Arria 10 and Intel Cyclone 10 GX designs, you can add the GENERATE_PR_RBF_FILE assignment to the .qsf or use the Convert Programming Files dialog box to convert the Partial-Masked SRAM Object Files (.pmsf) to an .rbf file, as Generating PR Bitstream Files on page 24 describes.

Figure 18. Programming File Generation

Base revision persona A
PR revision persona B
PR revision persona C

Partial Reconfiguration Design

Revision B

Revision C

Base Revision with Persona A

pr_region.pmsf
static.msf
base.sof

B.sof
B.pmsf

C.sof
C.pmsf

Note: When you generate a static .qdb for import into a PR implementation compile, make sure to preserve the entity-bound .sdc files for the static partition. Also, for the implementation revision to properly process the .sdc files, the order of assignments in the implementation file .qsf is very important. Verify the order of the .sdc files in the implementation revision. The implementation revision includes the entity-bound .sdc constraints pulled in by the static region .qdb. The implementation revision also includes the .sdc files for the implementation revision. If you require the .sdc files pulled in by the static region .qdb before the implementation revision .sdc files, ensure that the QDB_FILE_PARTITION assignment appears before any other .sdc file assignment.
### Table 4. PR Programming Files

<table>
<thead>
<tr>
<th>Programming File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;rev&gt;.&lt;pr_region&gt;.pmsf</code></td>
<td>Contains the partial-mask bits for the PR region. The .pmsf file contains all the information for creating PR bitstreams. Note: The default file name corresponds to the partition name.</td>
</tr>
<tr>
<td><code>&lt;rev&gt;.&lt;static_region&gt;.msf</code></td>
<td>Contains the mask bits for the static region.</td>
</tr>
<tr>
<td><code>&lt;rev&gt;.sof</code></td>
<td>Contains configuration information for the entire device.</td>
</tr>
</tbody>
</table>

**Related Information**
- PR Bitstream Security Verification (Intel Stratix 10 Designs) on page 56
- Intel Agilex Configuration User Guide
- Intel Stratix 10 Configuration User Guide
- Intel Arria 10 Configuration User Guide
- Intel Cyclone 10 GX Core Fabric and General Purpose I/Os Handbook

### 1.5.9.1. Generating PR Bitstream Files

For Intel Stratix 10 designs, the Assembler generates a configuration .rbf automatically at the end of compilation. For Intel Arria 10 and Intel Cyclone 10 GX designs, use any of the following methods to process the PR bitstreams and generate the Raw Binary File (.rbf) file for reconfiguration.

**Note:** The Intel Quartus Prime Pro Edition software supports compilation of PR designs for Intel Agilex devices, but does not yet support Intel Agilex PR bitstream generation.

#### Generating PR Bitstreams During Compilation

Follow these steps to generate the .rbf file during compilation:

1. Add the following assignments to the revision .qsf to automatically generate the required PR bitstreams following compilation:

   ```
   set_global_assignment -name GENERATE_PR_RBF_FILE ON
   set_global_assignment -name ON_CHIP_BITSTREAM_DECOMPRESSION OFF
   ```

2. To compile the revision and generate the .rbf, click **Processing ➤ Start Compilation**.

#### Generating PR Bitstreams with Programming File Generator

Follow these steps to generate the .rbf for PR programming with the **Programming File Generator**:

1. Click **File ➤ Programming File Generator**. The **Programming File Generator** appears.

2. Specify the target **Device family** and the **Configuration mode** for partial reconfiguration.

3. On the **Output File** tab, specify the **Output directory**, file name, and enable the **Raw Binary File for Partial Reconfiguration (.rbf)** file type.
4. To add the input .pmsf file to convert, click the Input Files tab, click Add Bitstream, and specify the .pmsf that you generated in the Assembler.

**Figure 19.** Adding Bitstream File

5. On the Input Files tab, select the bitstream .pmsf file and click Properties. Specify any of the following options for the .rbf:
   - **Enable compression**—generates compressed PR bitstream files to reduce file size.
   - **Enable encryption**—generates encrypted independent bitstreams for base image and PR image. You can encrypt the PR image even if your base image has no encryption. The PR image can have a separate encryption key file (.ekp). You can also specify other Security settings.
   - If you turn on **Enable encryption**, you must also acknowledge the Design Security Feature Disclaimer by checking the box.

**Figure 20.** Design Security Feature Disclaimer

6. Click OK.

7. In Programming File Generator, click Generate. The PR bitstream files generate according to your specifications.
Generating PR Bitstreams with Convert Programming Files Dialog Box

Follow these steps to generate the .rbf with the Convert Programming Files dialog box:

1. Click File ➤ Convert Programming Files. The Convert Programming Files dialog box appears.
2. Specify the output file name and Programming file type as Raw Binary File for Partial Reconfiguration (.rbf).
3. To add the input .pmsf file to convert, click Add File.
4. Select the newly added .pmsf file, and click Properties.
5. Enable or disable any of the following options and click OK:
   • Compression—enables compression on PR bitstream.
   • Enhanced compression—enables enhanced compression on PR bitstream.
   • Generate encrypted bitstream—generates encrypted independent bitstreams for base image and PR image. You can encrypt the PR image even if your base image has no encryption. The PR image can have a separate encryption key file (.ekp). If you enable Generate encrypted bitstream, enable or disable the Enable volatile security key, Use encryption lock file, and Generate key programming file options.
6. Click Generate. The PR bitstream files generate according to your specifications.
1.5.9.2. Partial Reconfiguration Bitstream Compatibility Checking

Partial reconfiguration bitstream compatibility checking verifies the compatibility of the reconfiguration bitstream to prevent configuration with an incompatible PR bitstream. The following sections describe PR bitstream compatibility check support.

Figure 21. PMSF File Properties Bitstream Encryption

Figure 22. PR Bitstream Compatibility Checking
**Intel Stratix 10 and Intel Agilex PR Bitstream Compatibility Checking**

For Intel Stratix 10 and Intel Agilex designs, PR bitstream compatibility checking is automatically enabled in the Compiler and in the Secure Device Manager (SDM) firmware by default. The following limitations apply to PR designs if PR bitstream compatibility checking is enabled:

- The firmware allows up to a total of 32 PR regions, irrespective of the number of hierarchical partial reconfiguration layers.
- Your PR design can have up to six hierarchical partial reconfiguration layers.
- Your PR design, when there is no hierarchy, can have up to 32 regions.
- Your PR design can have up to 15 child PR regions of any parent PR region (if it is hierarchical). Child PR regions count towards the total limit of 32 PR regions.

The Compiler generates an error if your PR design exceeds these limits when PR bitstream compatibility checking is enabled.

If you require more PR regions than this limitation allows, or otherwise want to disable PR bitstream compatibility checking, you can add the following assignment to the .qsf file:

```plaintext
set_global_assignment -name ENABLE_PR_POF_ID OFF
```

When you set this assignment to off, the limit of 32 total regions does not apply in the Compiler.

**Note:**

If you require the PR bitstream authentication feature for your design, you must enable PR bitstream compatibility checking by setting the global assignment ENABLE_PR_POF_ID to ON. The default setting is ON.

**Intel Arria 10 and Intel Cyclone 10 GX PR Bitstream Compatibility Checking**

For Intel Arria 10 and Intel Cyclone 10 GX designs, you enable or disable PR bitstream compatibility checking by turning on the **Enable bitstream compatibility check** option when instantiating the Partial Reconfiguration Controller Intel Arria 10/Cyclone 10 FPGA IP from the IP Catalog.

The PR IP verifies the partial reconfiguration PR Bitstream file (.rbf). When you enable the bitstream compatibility check, the PR .pof ID is encoded as the 71st word of the PR bitstream. If the PR IP detects an incompatible bitstream, then the PR IP stops the PR operation, and the status output reports an error.

When you turn on **Enable bitstream compatibility check**, the PR Controller IP core creates a **PR bitstream ID** and displays the bitstream ID in the configuration dialog box. For bitstream compatibility checking with hierarchical PR designs, refer to additional steps in **AN 806: Hierarchical Partial Reconfiguration Tutorial for Intel Arria 10 GX FPGA Development Board**.

**Related Information**

**AN 806: Hierarchical Partial Reconfiguration Tutorial for Intel Arria 10 GX FPGA Development Board**
1.5.9.3. Raw Binary Programming File Byte Sequence Transmission Examples

The raw binary programming file (.rbf) file contains the device configuration data in little-endian raw binary format. The following example shows transmitting the .rbf byte sequence 02 1B EE 01 in x32 mode:

<table>
<thead>
<tr>
<th>Table 5. Writing to the PR control block or SDM in x32 mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>In x32 mode, the first byte in the file is the least significant byte of the configuration double word, and the fourth byte is the most significant byte.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Double Word = 01EE1B02</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSB: BYTE0 = 02</td>
</tr>
<tr>
<td>BYTE1 = 1B</td>
</tr>
<tr>
<td>BYTE2 = EE</td>
</tr>
<tr>
<td>MSB: BYTE3 = 01</td>
</tr>
<tr>
<td>D[7..0]</td>
</tr>
<tr>
<td>D[15..8]</td>
</tr>
<tr>
<td>D[23..16]</td>
</tr>
<tr>
<td>D[31..24]</td>
</tr>
<tr>
<td>0000 0010</td>
</tr>
<tr>
<td>0001 1011</td>
</tr>
<tr>
<td>1110 1110</td>
</tr>
<tr>
<td>0000 0001</td>
</tr>
</tbody>
</table>

1.5.9.4. Generating a Merged .pmsf File from Multiple .pmsf Files

Use a single merged .rbf file to reconfigure two PR regions simultaneously. To merge two or more .pmsf files:

1. Open the Convert Programming Files dialog box.
2. Specify the output file name and programming file type as Merged Partial-Mask SRAM Object File (.pmsf).
3. In the Input files to convert dialog box, select PMSF Data.
4. To add input files, click Add File. You must specify two or more files for merging.
5. To generate the merged file, click Generate.

Alternatively, to merge two or more .pmsf files from the Intel Quartus Prime shell, type the following command:

```
quartus_cpf --merge_pmsf=<number of merged files> <pmsf_input_file_1> <pmsf_input_file_2> <pmsf_input_file_etc> <pmsf_output_file>
```

For example, to merge two .pmsf files, type the following command:

```
quartus_cpf --merge_pmsf=2 <pmsf_input_file_1> <pmsf_input_file_2> <pmsf_output_file>
```

1.6. Partial Reconfiguration Design Considerations

Partial reconfiguration is an advanced design flow in the Intel Quartus Prime Pro Edition software. Creating a partial reconfiguration design requires an understanding of how the PR design guidelines apply to your design. When designing for partial reconfiguration, you must consider the entire system-level behavior initial conditions to maintain the integrity and correctness of the static region operation.

For example, during PR programming, you must ensure that other parts of the system do not read or write to the PR region. You must also freeze the write enable output from the PR region into the static region, to avoid interference with static region operation. If all personas for your design do not have identical top-level interfaces, you must create the wrapper logic to ensure that all the personas appear similar to the static region. Upon partial reconfiguration of a PR region, you must bring the registers in the PR region to a known state by applying a reset sequence. There are
specific guidelines for global signals and on-chip memories. The following sections provide design considerations and guidelines to help you create design files for a PR design.

**FPGA Device and Software Considerations**

- Only Intel Arria 10 and Intel Cyclone 10 GX devices in -1, -2 and -3 speed grade support partial reconfiguration. All Intel Agilex and Intel Stratix 10 devices support PR.
- Use the nominal VCC of 0.9V or 0.95V as per the datasheet, including VID enabled devices.
- To minimize Intel Arria 10 and Intel Cyclone 10 GX programming files size, ensure that the PR regions are short and wide. For Intel Agilex and Intel Stratix 10 designs, use sector-aligned PR regions.
- The Intel Quartus Prime Standard Edition software does not support partial reconfiguration for Intel Arria 10 devices, nor provide any support for Intel Agilex nor Intel Stratix 10 devices.
- The current version of the Intel Quartus Prime Pro Edition software supports only one Signal Tap File (.stp) per revision.

**Design Partition Considerations**

- Reconfigurable partitions can only contain core resources, such as LABs, RAMs, and DSPs. All periphery resources, such as the transceivers, external memory interface, HPS, and clocks must be in the static portion of the design.
- To physically partition the device between static and individual PR regions, floorplan each PR region into exclusive, core-only, placement regions, with associated routing regions.
- A reconfiguration partition must contain the super-set of all ports that you use across all PR personas.

**Clocking, Reset, and Freeze Signal Considerations**

- The maximum number of clocks or other global signals for any Intel Arria 10 or Intel Cyclone 10 GX PR region is 33. The maximum number of clocks or other global signals for any Intel Agilex or Intel Stratix 10 PR region is 32. In the current version of the Intel Quartus Prime Pro Edition software, no two PR regions can share a row-clock.
- PR regions do not require any input freeze logic. However, you must freeze all the outputs of each PR region to a known constant value to avoid unknown data during partial reconfiguration.
- Increase the reset length by 1 cycle to account for register duplication in the Fitter.
- Ensure that all low-skew global signals (clocks and resets) driving into PR region in base revision compilations have destinations.
### 1.6.1. Partial Reconfiguration Design Guidelines

The following table lists important design guidelines at various steps in the PR design flow:

<table>
<thead>
<tr>
<th>PR Design Step</th>
<th>Guideline</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>Designing for partial reconfiguration</td>
<td>Do not assume initial states in registers inside PR region. After PR is complete, ensure that you reset all the control path registers to a known state, but omit the data path registers.</td>
<td>Registers inside the PR region contain undefined values after reconfiguration. Omitting data path registers reduces congestion on reset signals.</td>
</tr>
<tr>
<td></td>
<td>You cannot define synchronous reset as a global signal for Intel Arria 10 or Intel Cyclone 10 GX partial reconfiguration.</td>
<td>PR regions do not support synchronous reset of registers as a global signal, because the Intel Arria 10 and Intel Cyclone 10 GX LAB does not support synchronous clear (sclr) signal on a global buffer. The LAB supports the asynchronous clear (aclr) signal driven from a local input, or from a global network row clock. As a result, only the aclr can be a global signal, feeding registers in a PR region.</td>
</tr>
<tr>
<td></td>
<td>The PRESERVE_FANOUT_FREE_NODE assignment cannot preserve a fanout-free register that has no fanout inside the Verilog HDL or VHDL module in which you define it. To preserve these fanout-free registers, implement the noprune pragma in the source file: (<em>noprune</em>)reg r; If there are multiple instances of this module, with only some instances requiring preservation of the fanout-free register, set a dummy pragma on the register in the HDL and also set the PRESERVE_FANOUT_FREE_NODE assignment. This dummy pragma allows the register synthesis to implement the assignment. For example, set the following dummy pragma for a register r in Verilog HDL: (<em>dummy</em>)reg r; Then set this instance assignment: set_instance_assignment -name PRESERVE_FANOUT_FREE_NODE ON -to r;</td>
<td>The PRESERVE_FANOUT_FREE_NODE assignment does not apply when a register is not used in the Verilog HDL or VHDL module in which it is defined.</td>
</tr>
<tr>
<td>Partitioning the design</td>
<td>Register all the inputs and outputs for your PR region.</td>
<td>Improves timing closure and time budgeting.</td>
</tr>
<tr>
<td></td>
<td>Reduce the number of signals interfacing the PR region with the static region in your design.</td>
<td>Reduces the wire LUT count.</td>
</tr>
<tr>
<td></td>
<td>Create a wrapper for your PR region.</td>
<td>The wrapper creates common footprint to static region.</td>
</tr>
<tr>
<td></td>
<td>Drive all the PR region output ports to inactive state when the PR region is held in reset and the freeze bit is asserted for the PR region.</td>
<td>Prevents the static region logic from receiving random data during the partial reconfiguration operation.</td>
</tr>
</tbody>
</table>

*continued...*
1.6.2. PR Design Timing Closure Best Practices

The use of partition boundary ports for PR regions can make timing closure more challenging because the Compiler cannot optimize the logic across a partition boundary. The use of Logic Lock regions can also limit placement and routing flexibility. You must register all PR region boundary ports. Even when taking these steps, you may still find timing criticalities.

Each persona of a PR region can have different bits or input and output buses in use. Therefore, it is important to preserve the registers that do not have fan-out in a given persona, or that are driven by constants. You must ensure that the Compiler does not optimize away such registers during the compilation of a persona.

If the base compile does not use some bits of a bus, and the Compiler removes the corresponding registers for those bits, the logic may be untimed, resulting in unfavorable placement and routing. If you use those unregistered paths in other persona logic, you can have difficulty meeting timing on those paths. Preserving the unused port registers in the base compile ensures that the paths are timed in the base compile, and eases timing closure during persona compiles.

Follow these guidelines for effective register preservation in PR designs:

- Only the registers within PR regions require preservation.
- Only the PR base compilation requires register preservation.
- In a persona compile, the Compiler can safely remove fan-out free and constant-driven registers.
- For hierarchical PR compilations, only the base compile of the hierarchy requires register preservation.
- Preserve fan-out free nodes for input registers.
• Preserve constant-driven nodes for output registers.
• Only assign the attributes for the PR base compile. Remove the attributes for the persona compile (for example, via parameter or generic).
• You can set top-level parameters in the .qsf, which in turn pass down to lower hierarchies.

Use any of the following synthesis attributes to preserve registers:

• To preserve constant-driven or fan-out free registers, use the no_prune attribute. no_prune also disables all physical optimizations:

  Verilog: (* no_prune *) reg reg1;
  VHDL: signal reg1: std_logic;
        attribute no_prune: boolean;
        attribute no_prune of reg1: signal is true;

• To preserve fan-out free registers while allowing retiming on bits that have fan-outs, assign PRESERVE_FANOUT_FREE_NODE ON as altera_attribute:

  Verilog: (* altera_attribute = "-name PRESERVE_FANOUT_FREE_NODE ON" *) \
          reg reg1;
  VHDL: signal reg1: stdlogic;
        attribute altera_attribute : string;
        attribute altera_attribute of reg1: signal is "-name
        PRESERVE_FANOUT_FREE_NODE ON";

• Alternatively, use the dummy attribute with the PRESERVE_FANOUT_FREE_NODE ON assignment in the .qsf:

  Verilog: (* dummy *) reg reg1;
  VHDL: signal reg1: std_logic;
        attribute dummy: boolean;
        attribute dummy of reg1: signal is true;

  .qsf Assignment:

  set_instance_assignment -name PRESERVE_FANOUT_FREE_NODE ON \
   -to <hierarchical path to reg1>

• To preserve constant-driven registers while allowing retiming on bits that have drivers, use the preserve_syn_only attribute:

  Verilog: (* preserve_syn_only *) reg reg1;
  VHDL: signal reg1: std_logic;
        attribute preserve_syn_only : boolean;
        attribute preserve_syn_only of reg1: signal is true;

The following example shows how to assign attributes in PR base compile using a parameter in System Verilog and in VHDL. The example contains a parameter called base_compile, which is set to true for the PR base compile only.

System Verilog:
localparam ON_OFF_STRING = base_compile ? "ON": "OFF";
(* altera_attribute = {"-name PRESERVE_FANOUT_FREE_NODE ", ON_OFF_STRING} *)
logic [WIDTH-1:0] pr_input_register;
(* altera_attribute = {"-name PRESERVE_REGISTER_SYN_ONLY ", ON_OFF_STRING} *)
logic [WIDTH-1:0] pr_output_register;

VHDL:
attribute altera_attribute : string;
type attributeStr_type is array(boolean) of string(1 to 35);
constant attributeStr : attributeStr_type := (true => "-name
PRESERVE_FANOUT_FREE_NODE ON \

1.6.3. PR File Management

You can simplify the management of PR personas and their corresponding source files by observing one of the following PR project file management methods.

To illustrate these methods, consider a design that includes two PR regions, each with the possible apple, orange, and banana personas.

Figure 23. Example Design with Two PR Regions and Three Personas

Method 1 (Preferred): Specify Unique Entity and File Names for Each Persona

In PR file management method 1, you specify unique entity and file name pairs for each persona in the project. For example:

- Define the apple persona in the apple.sv file
- Define the orange persona in the orange.sv file
- Define the banana persona in the banana.sv file

Note: For successful compilation and PR operation, all personas must have the exact same port names and widths defined in each .sv file.
In the base PR revision RTL, you specify "apple" as the PR persona for both PR regions:

**Figure 24. Setting the Base PR Persona to "apple"**

When you set the base persona to [apple, apple] by setting u_fruit_0 and u_fruit_1 as the PR partition and regions, you can easily change the persona occupying the PR region using the **Entity Rebinding** (ENTITY_REBINDING) option in the Design Partitions Window, or by editing the .qsf directly, as the following examples show:

To specify the orange persona for the PR implementation (impl) revision:

```
set_instance_assignment -name ENTITY_REBINDING orange -to u_fruit_0
set_instance_assignment -name ENTITY_REBINDING orange -to u_fruit_1
```

To specify the banana persona for the PR implementation (impl) revision:

```
set_instance_assignment -name ENTITY_REBINDING banana -to u_fruit_0
set_instance_assignment -name ENTITY_REBINDING banana -to u_fruit_1
```

To specify the different personas for each PR region:

```
set_instance_assignment -name ENTITY_REBINDING orange -to u_fruit_0
set_instance_assignment -name ENTITY_REBINDING orange -to u_fruit_1
```

For each implementation revision, you must ensure that you include the corresponding source file in the project (*Project ➤ Add/Remove Files in Project*). 

**Method 2: Parametrize a Single File as PR Persona**

In PR file management method 2, you use a wrapper file and parameters to parameterize a single file that defines all personas. For example, consider the same design that method 1 describes with two PR regions, each with three possible personas for each PR region.
Figure 25. **Wrapper and Parameters in Single File**

In the following example, u_fruit_0 and u_fruit_1 are set as the PR partitions and regions in the base compile. The FRUIT_TYPE parameter of 0 generates the apple entity for both PR personas.

You can then change the parameter values to change the personas. For example, to specify the orange persona for both PR regions, set the FRUIT_TYPE parameter to 1: (FRUIT_TYPE=1)

In addition to these changes to RTL, you must also follow these additional steps when using method 2 to update the .qsf:

1. Create a copy of your fruit.sv file. Name the file with a unique name, such as x_fruit.sv. Also rename the entity to match the .sv file.
2. Set the default parameter of FRUIT_TYPE to a Verilog macro, in this case, `X_FRUIT_TYPE.`

Figure 26. **Copy of fruit.sv Saved as x_fruit.sv**

3. Add x_fruit.sv to the project (Project ➤ Add/Remove Files in Project).
4. Specify the following in the PR implementation revision’s .qsf file:
1. Creating a Partial Reconfiguration Design

a. Add the following line to set the `X_FRUIT_TYPE` Verilog macro to 1, and to specify the proper parameter for "fruit" that instantiates the correct `FRUIT_TYPE`:

```verilog
define X_FRUIT_TYPE 1
```

b. Specify the entity rebinding assignment to associate the new `x_fruit` entity with instances of `u_fruit_0` and `u_fruit_1`:

```verilog
set_instance_assignment -name ENTITY_REBINDING x_fruit -to u_fruit_0
set_instance_assignment -name ENTITY_REBINDING x_fruit -to u_fruit_1
```

These `.qsf` changes set orange as the new persona for both PR regions.

5. One downside of method 2 is that the Verilog macro set in the `.qsf` file is global. Therefore, every instance of `X_FRUIT_TYPE` in the project defaults to a value of 1. This result may be suitable if you want both PR personas to be of the same type. However, if you want to specify "orange" for one persona and "banana" for another persona in one PR implementation compile, you must create another copy of the `fruit.sv` file with a unique name and Verilog macro:

![Copy of x_fruit.sv Saved as y_fruit.sv](image)

6. Specify the following in the PR implementation revision's `.qsf` file:

a. Add the following line to set the `X_FRUIT_TYPE` Verilog macro to 1, and to specify the proper parameter for "fruit" that instantiates the correct `FRUIT_TYPE`:

```verilog
define X_FRUIT_TYPE 1
```

b. Specify the entity rebinding assignment to associate the new `x_fruit` entity with instances of `u_fruit_0` and the new `y_fruit` entity with `u_fruit_1`:

```verilog
set_instance_assignment -name ENTITY_REBINDING x_fruit -to u_fruit_0
set_instance_assignment -name ENTITY_REBINDING x_fruit -to u_fruit_1
set_instance_assignment -name ENTITY_REBINDING y_fruit -to u_fruit_1
```

These `.qsf` changes specify orange as the persona for the first PR region, and banana as the persona for the second PR region.
1.6.4. Evaluating PR Region Initial Conditions

Unintended initial conditions in a PR region can lead to errors during partial reconfiguration. Your design may include unintended initial conditions, especially if you port a design not originally intended for partial reconfiguration. The Intel Quartus Prime Pro Edition software reports any initial conditions in the PR partitions for your evaluation following synthesis.

After compiling the base revision that defines the partition, you can view the Registers with Explicit Power-Up Settings report for the partition to identify, locate, and correct any unintended initial conditions. For a specific PR partition, you can view the power-up initial values after synthesizing the base revision in the Synthesis report. The Synthesis report includes power-up initial values in the Partition Statistics section.

![Partition Statistics in Synthesis Report](image)

The Messages window also generates a warning or error message about any initial conditions during synthesis processing. After evaluating the initial condition, you can determine whether the condition is correct for design functionality, or change the design to remove dependence on an initial condition that is incompatible with partial reconfiguration.

1.6.5. Creating Wrapper Logic for PR Regions

If all personas for your design do not have identical top-level interfaces, you must create the wrapper logic to ensure that all the personas appear similar to the static region. Define a wrapper for each persona, and instantiate the persona logic within the wrapper. If all personas have identical top-level interfaces, the personas do not require wrapper logic. In this wrapper, you can create dummy ports to ensure that all the personas of a PR region have the same connection to the static region.

During the PR compilation, the Compiler converts each of the non-global ports on interfaces of the PR region into boundary port wire LUTs. The naming convention for boundary port wire LUTs are `<input_port>~IPORT` for input ports, and `<output_port>~OPORT` for output ports. For example, the instance name of the wire LUT for an input port with the name `my_input`, on a PR region with the name `my_region`, is `my_region|my_input~IPORT`.

1. Manually floorplan the boundary ports using Logic Lock region assignments, or place the boundary ports automatically using the Fitter. The Fitter places the boundary ports during the base revision compile. The boundary LUTs are invariant locations the Fitter derives from the persona you compile. These LUTs represent
the boundaries between the static region and the PR routing and logic. The placement remains stationary regardless of the underlying persona, because the routing from the static logic does not vary with a different persona implementation.

2. To constrain all boundary ports within a given region, use a wildcard assignment. For example:

```
set_instance_assignment -name PLACE_REGION "65 59 65 85" -to \
  u_my_top|design_inst|pr_inst|pr_inputs.data_in*~IPORT
```

This assignment constrains all the wire LUTS corresponding to the IPORTS that you specify within the place region, between the coordinates (65 59) and (65 85).

**Figure 29. Wire-LUTs at the PR Region Boundary**

Optionally, floorplan the boundary ports down to the LAB level, or individual LUT level. To floorplan to the LAB level, create a 1x1 Logic Lock PLACE_REGION constraint (single LAB tall and a single LAB wide). Optionally, specify a range constraint by creating a Logic Lock placement region that spans the range. For more information about floorplan assignments, refer to *Floorplan the Partial Reconfiguration Design*.

**Related Information**

Step 3: Floorplan the Design on page 13

For more information on floorplanning your design.

### 1.6.6. Creating Freeze Logic for PR Regions

When partially reconfiguring a design, freeze all the outputs of each PR region to a known constant value. This freezing prevents the signal receivers in the static region from receiving undefined signals during the partial reconfiguration process.

The PR region cannot drive valid data until the partial reconfiguration process is complete, and the PR region is reset. Freezing is important for control signals that you drive from the PR region.

The freeze technique that you choose is optional, depending on the particular characteristics of your design. The freeze logic must reside in the static region of your design. A common freeze technique is to instantiate 2-to-1 multiplexers on each output of the PR region, to hold the output constant during partial reconfiguration.

**Note:** There is no requirement to freeze the global and non-global inputs of a PR region.
An alternative freeze technique is to register all outputs of the PR region in the static region. Then, use an enable signal to hold the output of these registers constant during partial reconfiguration.

The Partial Reconfiguration Region Controller IP core includes a freeze port for the region that it controls. Include this IP component with your system-level control logic to freeze the PR region output. For designs with multiple PR regions, instantiate one PR Region Controller IP core for each PR region in the design. The Intel Quartus Prime software includes the Avalon Memory-Mapped Freeze Bridge and Avalon Streaming Freeze Bridge Intel FPGA IP cores. You can use these IP cores to implement freeze logic, or design your own freeze logic for these standard interface types.

The static region logic must be independent of all the outputs from the PR regions for a continuous operation. Control the outputs of the PR regions by adding the appropriate freeze logic for your design.

1.6.7. Resetting the PR Region Registers

Upon partial reconfiguration of a PR region, the status of the PR region registers become indeterminate. Bring the registers in the PR region to a known state by applying a reset sequence for the PR region. This reset ensures that the system
behaves to your specifications. Simply reset the control path of the PR region, if the
datapath eventually flushes out within a finite number of cycles. Use active-high local
reset instead of active-low, wherever applicable. This technique allows you to
automatically hold the PR region in reset, by virtue of the boundary port wire LUT.

Table 7. Supported PR Reset Implementation Guideline

<table>
<thead>
<tr>
<th>PR Reset Type</th>
<th>Active-High Synchronous Reset</th>
<th>Active-High Asynchronous Reset</th>
<th>Active-Low Synchronous Reset</th>
<th>Active-Low Asynchronous Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>On local signal</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>On global signal</td>
<td>• No (Intel Arria 10)</td>
<td>• No (Intel Cyclone 10 GX)</td>
<td>• No (Intel Arria 10)</td>
<td>Yes (Intel Stratix 10)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1.6.8. Promoting Global Signals in a PR Region

In non-PR designs, the Intel Quartus Prime software automatically promotes high fan-
out signals onto dedicated global networks. The global promotion occurs in the Plan
stage of design compilation.

In PR designs, the Compiler disables global promotion for signals originating within the
logic of a PR region. Instantiate the clock control blocks only in the static region,
because the clock floorplan and the clock buffers must be a part of the static region of
the design. Manually instantiating a clock control block in a PR region, or assigning a
signal in a PR region with the GLOBAL_SIGNAL assignment, results in compilation
error. To drive a signal originating from the PR region onto a global network:

1. Expose the signal from the PR region.
2. Drive the signal onto the global network from the static region.
3. Drive the signal back into the PR region.

You can drive a maximum of 33 clocks (for Intel Arria 10 and Intel Cyclone 10 GX
devices), or 32 clocks (for Intel Agilex and Intel Stratix 10 devices) into any PR region. You
cannot share a row clock between two PR regions.

The Compiler allows only certain signals to be global inside a PR region. Use only
global signals to route secondary signals into a PR region, as the following table
describes:

Table 8. Supported Signal Types for Driving Clock Networks in a PR Region

<table>
<thead>
<tr>
<th>Block Type</th>
<th>Supported Global Network Signals</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAB, MLAB</td>
<td>Clock, ACLR, SCLR(4)</td>
</tr>
<tr>
<td>RAM, ROM (M20K)</td>
<td>Clock, ACLR, Write Enable (WE), Read Enable (RE), SCLR</td>
</tr>
<tr>
<td>DSP</td>
<td>Clock, ACLR, SCLR</td>
</tr>
</tbody>
</table>

(4) Only Intel Agilex and Intel Stratix 10 designs support global SCLR.
1.6.8.1. Viewing Row Clock Region Boundaries

You can use the Chip Planner to visualize the row clock region boundaries, and to ensure that no two PR regions share a row clock region.

1. Right-click a PR partition name in the Design Partitions Window and click Locate ▶ Locate in Chip Planner.

Figure 32. Row Clock Region Boundaries in Chip Planner

2. In Chip Planner, click the Layers tab and select the Basic layer. The Chip Planner overlays the row clock region boundaries. Adjust the Basic layer settings to display specific items.

1.6.9. Planning Clocks and other Global Routing

There are special PR considerations for the planning for clocks and other global routing. For Intel Agilex and Intel Stratix 10 designs, you can use the low skew networks (globals) for clocks or resets.

During the base revision compile, you must route any global signal that any PR persona requires into a destination in the PR region. For clocks signals, this destination is a register or other synchronous element and the signal entering the clock input. For a reset, the destination should be fed into the appropriate input.

This requirement occurs because PR only reconfigures the last part of the low skew network. If you do not route the root and middle sections of the network during the base compile, you cannot use that revision for the PR.
Consider an example with a super-set of signals for a PR region that consists of:

- Three clocks—clk_1, clk_2, and clk_3.
- Two resets—rst_1 and rst_2.
- Base PR persona—uses clk_1, clk_2, and rst_1 only.
- Other personas—use clk_3 and rst_2 only.

In this example, the base persona must have a proper destination for the "unused" clk_3 and rst_2. You can accomplish this by driving a single register with a (*no prune*) directive inside the base PR persona, with clk_3 and reset using rst_2.

Omitting these destinations results in an error during compilation of the PR implementation second persona.

1.6.10. Implementing Clock Enable for On-Chip Memories with Initialized Contents

Follow these guidelines to implement clock enable for on-chip memories with initialized contents:

1. To avoid spurious writes during PR programming for memories with initialized contents, implement the clock enable circuit in the same PR region as the M20K or MLAB RAM. This circuit depends on an active-high clear signal from the static region.

2. Before you begin the PR programming, assert this signal to disable the memory’s clock enable. Your system PR controller must deassert the clear signal on PR programming completion. You can use the freeze signal for this purpose.

3. Use the Intel Quartus Prime IP Catalog or Platform Designer to instantiate the On-Chip Memory and RAM Intel FPGA IP cores that include an option to automatically add this circuitry.

Figure 33. RAM Clock Enable Circuit for PR Region
Example 1.  Verilog RTL for Clock Enable

```verilog
reg ce_reg;
reg [1:0] ce_delay;
always @(posedge clock, posedge freeze) begin
  if (freeze) begin
    ce_delay <= 2'b0;
  end
  else begin
    ce_delay <= {ce_delay[0], 1'b1};
  end
end
always @(posedge clock, negedge ce_delay[1]) begin
  if (~ce_delay[1]) begin
    ce_reg <= 1'b0;
  end
  else begin
    ce_reg <= clken_in;
  end
end
wire ram_wrclocken;
assign ram_wrclocken = ce_reg;
```

Example 2.  VHDL RTL for Clock Enable

```vhdl
ENTITY mem_enable_vhd IS PORT(
  clock : in std_logic;
  freeze : in std_logic;
  clken_in : in std_logic;
  ram_wrclocken : out std_logic);
END mem_enable_vhd;
ARCHITECTURE behave OF mem_enable_vhd is
  SIGNAL ce_reg: std_logic;
  SIGNAL ce_delay: std_logic_vector(1 downto 0);
BEGIN
  PROCESS (clock, freeze)
  BEGIN
    IF ((clock'EVENT AND clock = '1') or (freeze'EVENT AND freeze = '1')) THEN
      IF (freeze = '1') THEN
        ce_delay <= "00";
      ELSE
        ce_delay <= ce_delay(0) & '1';
      END IF;
    END IF;
  END PROCESS;
  PROCESS (clock, ce_delay(1))
  BEGIN
    IF ((clock'EVENT AND clock = '1') or (ce_delay(1)'EVENT AND ce_delay(1) = '0')) THEN
      IF (ce_delay(1) = '0') THEN
        ce_reg <= '0';
      ELSE
        ce_reg <= clken_in;
      END IF;
    END IF;
  END PROCESS;
  ram_wrclocken <= ce_reg;
END ARCHITECTURE behave;
```
1.6.10.1. Clock Gating

An alternate method to avoid spurious writes of initialized content memories is to implement clock gating circuitry in the PR static region, and feed the clock gating circuitry to the PR region in which the initialized memories are implemented.

**Figure 34. Global Clock Control Block**

Implement the gating circuitry in the static region, and feed it to the PR region in which the initialized memories are being implemented. Clock gating is logically equivalent to using clock enable on the memories. This method provides the following benefits:

- Uses the enable port of the global clock buffers to disable the clock before starting the partial reconfiguration operation. Also enables the clock on PR completion.
- Ensures that the clock does not switch during reconfiguration, and requires no additional logic to avoid spurious writes.

**Related Information**

Clock Control Block (ALTCLKCTRL) Intel FPGA IP User Guide
1.7. Hierarchical Partial Reconfiguration

Hierarchical partial reconfiguration (HPR) is an extension of partial reconfiguration (PR), where you contain one PR region within another PR region. You can create multiple personas for both the child and parent partitions. You nest the child partitions within their parent partitions. Reconfiguring a parent partition does not impact the operation in the static region, but replaces the child partitions of the parent region with default child partition personas.

The HPR design flow includes the following steps:

1. Create a base revision for the design and export the static region, as Step 7: Compile the Base Revision and Export the Static Region on page 20 describes.

2. Create the implementation revision for each persona, as Step 8: Setup PR Implementation Revisions on page 22 describes, and export the parent partitions.

3. Specify the .qdb file partition for the static and parent regions.

4. Specify the corresponding entity for the parent or child.

When compiling the implementation revision for an HPR design, you must fully floorplan the child partition, similar to planning the PR region of a base revision. Refer to Using Parent QDB Files from Different Compiles on page 46.

Note: Hierarchical PR (HPR) designs do not support PR bitstream security verification.

Related Information

AN826: Hierarchical Partial Reconfiguration Tutorial for Intel Stratix 10 GX FPGA Development Board
for step-by-step HPR instructions

1.7.1. Using Parent QDB Files from Different Compiles

For HPR designs, you can use the parent .qdb file from the same or different implementation compiles. The following examples illustrate two possible HPR compilation flows, with respect to the following design example block diagram:

Figure 35. Example HPR Design with Parent QDB Files from Different Compiles

![Example HPR Design with Parent QDB Files from Different Compiles]
Figure 35 on page 46 shows an HPR design hierarchy with the following characteristics:

- The blue and orange regions represent the HPR parent regions.
- The yellow and green boxes represent the child PR regions.
- The blue HPR parent has two personas, blue_1 and blue_2.
- For the yellow child region, the default persona that is compiled with parent blue_1 is yellow_1_1.
- The second child persona that can be compiled by blue_1 is yellow_2_1.
- The orange HPR parent has the same characteristics as the blue HPR parent.

Considering these HPR design characteristics, the following describes one possible HPR compilation flow:

**HPR Compilation Flow A:**

1. Blue_1, yellow_1_1, orange_1, green_1_1.
2. Blue_1.qdb, yellow_1_2, orange_1.qdb, green_1_2
3. Blue_2, yellow_2_1, orange_2, green_2_1
4. Blue_2.qdb, yellow_2_2, orange_2.qdb, green_2_2

In Flow A, in steps 2 and 4, the parent region .qdb files come from the same implementation compile. Step 2 uses blue_1.qdb and orange_1.qdb that step 1 generates in the same implementation compile.

HPR also supports import of the parent and child PR partitions from different implementation compiles:

**HPR Compilation Flow B:**

1. Blue_1, yellow_1_1, orange_1, green_1_1.
2. Blue_2, yellow_2_1, orange_2, green_2_1
3. Blue_1.qdb, yellow_1_2, orange_2.qdb, green_2_2
4. Blue_2.qdb, yellow_2_2, orange_1.qdb, green_1_2

In Flow B, blue_1.qdb and orange_2.qdb come from two different implementation compiles. Step 1 generates blue_1.qdb. Step 2 implementation compile generates orange_2.qdb.

**1.8. Partial Reconfiguration Design Timing Analysis**

The interface between partial and static partitions remains the same for each PR implementation revision. Perform timing analysis on each PR implementation revision to ensure that there are no timing violations. To ensure timing closure of a design with multiple PR regions, you can create aggregate revisions for all possible PR region combinations for timing analysis.
Note: Logic Lock regions impose placement constraints that affect the performance and resource utilization of your PR design. Ensure that the design has additional timing allowance and available device resources. Selecting the largest and most timing-critical persona as your base persona optimizes the timing closure. In addition, if you compile the base design with time borrowing enabled, compile the implementation designs with time borrowing enabled. Otherwise, time borrowing amounts in the base design are reset to zero, and the design may not pass timing. If this condition occurs, you can use the `update_timing_netlist -recompute_borrow` command to restore time borrowing amounts throughout the design for timing analysis.

Related Information

1.8.1. Running Timing Analysis on Aggregate Revisions
To ensure timing closure of a design with multiple PR regions, you create aggregate revisions for all possible PR region combinations and run timing analysis.

1. To open the **Revisions** dialog box, click **Project ➤ Revisions**.
2. To create a new revision, double-click `<<new revision>>`.
3. Specify the **Revision name** and select the base revision for **Based on Revision**.
4. To export the post-fit database from the base compile (static partition), type the following command in the Intel Quartus Prime shell:

```bash
quartus_cdb <project name> <base revision> --export_block \
"root_partition" --snapshot final --file "<base revision name>.qdb"
```

**Note:** Ensure that you include all the `.sdc` and `.ip` files for the static and PR regions. To detect the clocks, ensure that the `.sdc` file for the PR Controller IP follows the entry of any `.sdc` file that creates the clocks that the IP core uses. You facilitate this order by ensuring the `.ip` file for the PR Controller IP comes after any `.ip` or `.sdc` files that you use to create these clocks in the `.qsf` file for the project revision. Refer to Partial Reconfiguration Solutions IP User Guide on page 69 for more information.

5. To export the post-fit database from multiple personas (for the PR implementation revisions), type the following commands in the Intel Quartus Prime shell:

```bash
quartus_cdb <project name> -c <PR1 revision> --export_block \
<PR1 Partition name> --snapshot final --file "pr1.qdb"
quartus_cdb <project name> -c <PR2 revision> --export_block \
<PR2 Partition name> --snapshot final --file "pr2.qdb"
```

6. To import the post-fit databases of the static region as an aggregate revision, type the following commands in the Intel Quartus Prime shell:

```bash
quartus_cdb <project name> -c <aggr_rev> --import_block \
"root_partition" --file "<base revision name>.qdb"
quartus_cdb <project name> -c <aggr_rev> --import_block \
<PR1 partition name> --file "pr1.qdb"
quartus_cdb <project name> -c <aggr_rev> --import_block \
<PR2 Partition name> --file "pr2.qdb"
```
7. To integrate post-fit database of all the partitions, type the following command in the Intel Quartus Prime shell:

```
quartus_fit <project name> -c <aggr_rev>
```

*Note:* The Fitter verifies the legality of the post-fit database, and combines the netlist for timing analysis. The Fitter does not reroute the design.

8. To perform timing analysis on the aggregate revision, type the following command in the Intel Quartus Prime shell:

```
quartus_sta <proj name> -c <aggr_rev>
```

9. Run timing analysis on aggregate revision for all possible PR persona combinations. If a specific persona fails timing closure, recompile the persona and perform timing analysis again.

### 1.9. Partial Reconfiguration Design Simulation

Simulation verifies the behavior of your design before device programming. The Intel Quartus Prime Pro Edition software supports simulating the delivery of a partial reconfiguration bitstream to the PR control block. This simulation allows you to observe the resulting change and the intermediate effect in a reconfigurable partition.

The Intel Quartus Prime Pro Edition software supports simulation of PR persona transitions through the use of simulation multiplexers. You use the simulation multiplexers to change which persona drives logic inside the PR region during simulation. This simulation allows you to observe the resulting change and the intermediate effect in a reconfigurable partition.

Similar to non-PR design simulations, preparing for a PR simulation involves setting up your simulator working environment, compiling simulation model libraries, and running your simulation. The Intel Quartus Prime software provides simulation components to help simulate a PR design, and can generate the gate-level PR simulation models for each persona. Use either the behavioral RTL or the gate-level PR simulation model for simulation of the PR personas. The gate-level PR simulation model allows for accurate simulation of registers in your design and reset sequence verification. These technology-mapped registers do not assume initial conditions.

You can use the PR mode of the EDA netlist writer to generate the gate level netlist of a PR region. Refer to the "EDA Netlist Writer and Gate Level-Netlists" section of the Intel Quartus Prime Pro Edition User Guide: Third Party Simulation.

#### Related Information
- Generating and Simulating Intel FPGA IP on page 122

### 1.9.1. Partial Reconfiguration Simulation Flow

At a high-level, a PR operation consists of the following steps:

1. System-level preparation for a PR event.
2. Retrieval of the partial bitstream from memory.
3. Transmission of the partial bitstream to the PR control block or SDM.
4. Resulting change in the design as a new persona becomes active.
5. Post-PR system coordination.
6. Use of the new persona in the system.

You can simulate each of these process steps in isolation, or as a larger sequence depending on your verification type requirement.

**Related Information**
- Intel Arria 10 and Intel Cyclone 10 GX PR Control Block Simulation Model on page 128
- Generating the PR Persona Simulation Model on page 130

### 1.9.1.1. Simulating PR Persona Replacement

The logical operation of the PR partition changes when a new persona loads during the partial reconfiguration process. Simulate the replacement of personas using multiplexers on the input and output of the persona under simulation. Create RTL wrapper logic to represent the top-level of the persona. The wrapper instantiates the default persona during compilation. During simulation, the wrapper allows the replacement of the active persona with another persona. Instantiate each persona as the behavioral RTL in the PR simulation model the Intel Quartus Prime EDA Netlist Writer generates. The Intel Quartus Prime software includes simulation modules to interface with your simulation testbench:

- `altera_pr_wrapper_mux_in`
- `altera_pr_wrapper_mux_out`
- `altera_pr_persona_if` (SystemVerilog interface allows you to connect the wrapper multiplexers to a testbench driver)
Example 3. RTL Wrapper for PR Persona Switching Simulation

The `pr_activate` input of the `altera_pr_wrapper_mux_out` module enables the MUX to output X. This functionality allows the simulation of unknown outputs from the PR persona, and also verifies the normal operation of the design’s freeze logic. The following code corresponds to the simulation of PR persona switching, shown in the above figure:

```verilog
module pr_core_wrapper
(
    input wire a,
    input wire b,
    output wire o
);

localparam ENABLE_PERSONA_1 = 1;
localparam ENABLE_PERSONA_2 = 1;
localparam ENABLE_PERSONA_3 = 1;
localparam NUM_PERSONA = 3;

logic pr_activate;
int persona_select;

altera_pr_persona_if persona_bfm();
assign pr_activate = persona_bfm.pr_activate;
assign persona_select = persona_bfm.persona_select;

wire a_mux [NUM_PERSONA-1:0];
wire b_mux [NUM_PERSONA-1:0];
wire o_mux [NUM_PERSONA-1:0];

generate
    if (ENABLE_PERSONA_1)
        localparam persona_id = 0;
```
```
`ifdef ALTERA_ENABLE_PR_MODEL
    assign u_persona_0.altera_sim_pr_activate = pr_activate;
`endif

pr_and u_persona_0
{
    .a(a_mux[persona_id]),
    .b(b_mux[persona_id]),
    .o(o_mux[persona_id])
};
end
endgenerate

generate
if (ENABLE_PERSONA_2) begin
    localparam persona_id = 1;
    `ifdef ALTERA_ENABLE_PR_MODEL
        assign u_persona_1.altera_sim_pr_activate = pr_activate;
    `endif
    pr_or u_persona_1
{
    .a(a_mux[persona_id]),
    .b(b_mux[persona_id]),
    .o(o_mux[persona_id])
};
end
endgenerate

generate
if (ENABLE_PERSONA_3) begin
    localparam persona_id = 2;
    `ifdef ALTERA_ENABLE_PR_MODEL
        assign u_persona_2.altera_sim_pr_activate = pr_activate;
    `endif
    pr_empty u_persona_2
{
    .a(a_mux[persona_id]),
    .b(b_mux[persona_id]),
    .o(o_mux[persona_id])
};
end
endgenerate

altera_pr_wrapper_mux_in #(.NUM_PERSONA(NUM_PERSONA), .WIDTH(1))
    u_a_mux(.sel(persona_select), .mux_in(a), .mux_out(a_mux));

altera_pr_wrapper_mux_in #(.NUM_PERSONA(NUM_PERSONA), .WIDTH(1))
    u_b_mux(.sel(persona_select), .mux_in(b), .mux_out(b_mux));

altera_pr_wrapper_mux_out #(.NUM_PERSONA(NUM_PERSONA), .WIDTH(1))
    u_o_mux(.sel(persona_select), .mux_in(o_mux), .mux_out(o), .pr_activate
    (pr_activate));
endmodule
```

### 1.9.1.2. `altera_pr_persona_if` Module

Instantiate the `altera_pr_persona_if` SystemVerilog interface in a PR region simulation wrapper to connect to all the wrapper multiplexers. Optionally, connect `pr_activate` to the PR simulation model.
Connect the interface’s persona_select to the sel port of all input and output multiplexers. Connect the pr Activate to the pr_activate of all the output multiplexers. Optionally, connect the report events to the report event ports of the PR simulation model. Then, the PR region driver testbench component can drive the interface.

interface altera_pr_persona_if;
    logic pr_activate;
    int persona_select;
    event report_storage_if_x_event;
    event report_storage_if_1_event;
    event report_storage_if_0_event;
    event report_storage_event;

    initial begin
        pr_activate <= 1'b0;
    end
endinterface : altera_pr_persona_if

The \texttt{<QUARTUS\_INSTALL\_DIR>/eda/sim\_lib/altera\_lnsim.sv} file defines the \texttt{altera\_pr\_persona\_if} component.

1.9.1.3. \texttt{altera_pr_wrapper_mux_out} Module

The \texttt{altera_pr_wrapper_mux_out} module allows you to multiplex the outputs of all PR personas to the outputs of the PR region wrapper.

Instantiate one multiplexer per output port. Specify the active persona using the sel port of the multiplexer. The pr_activate port allows you to drive the multiplexer output to “x”, to emulate the unknown value of PR region outputs during a PR operation. Parameterize the component to specify the number of persona inputs, the multiplexer width, and the MUX output value when pr_activate asserts.

module altera_pr_wrapper_mux_out #(
    parameter NUM_PERSONA = 1,
    parameter WIDTH = 1,
    parameter [0:0] DISABLED_OUTPUT_VAL = 1'bx
) (

    input int sel,
    input wire [WIDTH-1 : 0] mux_in [NUM_PERSONA-1:0],
    output reg [WIDTH-1:0]   mux_out,
    input wire               pr_activate
);

    always_comb begin
        if (((sel < NUM_PERSONA) && (!pr_activate))
            mux_out = mux_in[sel];
        else
            mux_out = {WIDTH{DISABLED_OUTPUT_VAL}};
    end

endmodule : altera_pr_wrapper_mux_out

The \texttt{<QUARTUS\_INSTALL\_DIR>/eda/sim\_lib/altera\_lnsim.sv} file defines the \texttt{altera_pr_wrapper_mux_out} component.

1.9.1.4. \texttt{altera_pr_wrapper_mux_in} Module

The \texttt{altera_pr_wrapper_mux_in} module allows you to de-multiplex inputs to a PR partition wrapper for all PR personas.
Instantiate one multiplexer per input port. Specify the active persona using the sel port of the multiplexer. Parameterize the component to specify the number of persona outputs, the multiplexer width, and the MUX output for any disabled output. When using the `altera_pr_wrapper_mux_in` to mux a clock input, use the DISABLED_OUTPUT_VAL of 0, to ensure there are no simulation clock events of the disabled personas.

```verilog
module altera_pr_wrapper_mux_in#(
    parameter NUM_PERSONA = 1,
    parameter WIDTH = 1,
    parameter [0:0] DISABLED_OUTPUT_VAL = 1'bx
) (
    input int sel,
    input wire [WIDTH-1:0] mux_in,
    output reg [WIDTH-1 : 0] mux_out [NUM_PERSONA-1:0]
);
always_comb begin
    for (int i = 0; i < NUM_PERSONA; i++)
        if (i == sel)
            mux_out[i] = mux_in;
        else
            mux_out[i] = {WIDTH{DISABLED_OUTPUT_VAL}};
end
endmodule : altera_pr_wrapper_mux_in
```

The `<QUARTUS_INSTALL_DIR>/eda/sim_lib/altera_lnsim.sv` file defines the `altera_pr_wrapper_mux_in` component.

### 1.10. Partial Reconfiguration Design Debugging

The following Intel FPGA IP cores support system-level debugging in the static region of a PR design:

- In-System Memory Content Editor
- In-System Sources and Probes Editor
- Virtual JTAG
- Nios II JTAG Debug Module
- Signal Tap Logic Analyzer

In addition, the Signal Tap logic analyzer allows you to debug the static or partial reconfiguration (PR) regions of the design. If you only want to debug the static region, you can use the In-System Sources and Probes Editor, In-System Memory Content Editor, or System Console with the JTAG Avalon Master Bridge.

**Related Information**


#### 1.10.1. Debugging PR Designs with the Signal Tap Logic Analyzer

To use the Signal Tap logic analyzer to debug PR designs, you must create a debug bridge to extend Signal Tap debugging into the PR partition. You can then use Signal Tap to debug by connecting to the debug bridge. To use the debug bridge, you instantiate the SLD JTAG Bridge Agent Intel FPGA IP and SLD JTAG Bridge Host Intel FPGA IP pair for each PR region in your design.
Perform the following steps during the early planning stage, to ensure you can use Signal Tap to debug your static as well as PR region:

1. Instantiate the SLD JTAG Bridge Agent IP in the static region.
2. Instantiate the SLD JTAG Bridge Host IP in the PR region of the default persona.
3. Instantiate the SLD JTAG Bridge Host IP for each of the personas when creating revisions for the personas.

The Signal Tap logic analyzer uses the hierarchical debug capabilities provided by the Intel Quartus Prime software to tap signals in the static and PR regions simultaneously.

You can debug multiple personas present in your PR region, as well as multiple PR regions. For complete information on the debug infrastructure using hierarchical hubs, refer to Intel Quartus Prime Pro Edition User Guide: Debug Tools.

Related Information
- AN 845: Signal Tap Tutorial for Intel Arria 10 Partial Reconfiguration Design
- AN 841: Signal Tap Tutorial for Intel Stratix 10 Partial Reconfiguration Design
- Instantiating the SLD JTAG Bridge Agent
- Instantiating the SLD JTAG Bridge Host
1.11. PR Bitstream Security Verification (Intel Stratix 10 Designs)

Intel Stratix 10 devices support optional PR bitstream security verification to confirm that a PR region persona contains no threats to platform integrity or confidentiality before loading that persona into the FPGA.

When you license and enable PR bitstream security verification, the Programmer performs bit ownership, signal peek, poke, and contention checks prior to PR bitstream generation. If the Programmer cannot verify the PR persona `.pmsf` file against the Secure Mask Settings file (.smsf), PR bitstream generation terminates.

*Note:* PR bitstream security verification only supports Intel Stratix 10 devices. Hierarchical PR (HPR) designs do not support PR bitstream security verification. This feature requires a separate license and `.qsf` setting to enable, as Using PR Bitstream Security Verification (Intel Stratix 10 Designs) on page 57 describes.

PR bitstream security verification enables multi-tenant FPGA usage, such as when a Platform Owner partitions a single device to host multiple third-party Clients. The Platform Owner may not trust the Clients, and the Clients may not trust each other, but the Clients trust the Platform Owner. PR bitstream security verification provides the Platform Owner and Clients protection from any party corrupting the proprietary server, Client configurations, or from initiating a peek or poke attack by a subsequent partial reconfiguration.

In such multi-tenant scenarios, PR bitstream security verification can protect against the following known threats:

- **Changing bits in a PR region owned by another party**—a PR persona can potentially inject a change in functionality to another client’s persona and cause a DOS or PDOS attack, cause an unintended function, or redirect a signal from the unowned persona.
- **Poke Attack**—a PR persona can potentially corrupt other PR regions input data by poking on its wires.
- **Peek Attack**—a PR persona can potentially view values on routing wires from other PR regions to snoop on their data.
- **Excessive Power Consumption**—PR personas can potentially cause excessive FPGA power consumption due to wire contention inside the PR region.

1.11.1. PR Bitstream Security Use Case (Intel Stratix 10 Designs)

In a typical use case, PR bitstream security validation allows a Platform Owner to verify a Client’s third-party IP design to ensure that it contains no known threats to platform integrity and confidentiality before that PR persona can be loaded into an FPGA during partial reconfiguration. This validation is critical for Cloud Service Providers and system integrators.

A Cloud Service Provider partitions the FPGA into various PR regions to host multiple third-party clients. The Cloud Service Provider uses this partitioned design for the base device configuration. After the base configuration loads on the device, PR bitstream security allows only verified PR region persona to load into the corresponding PR region.
1.11.2. Using PR Bitstream Security Verification (Intel Stratix 10 Designs)

PR bitstream security verification requires a separate license and `.qsf` setting to enable. After you license and enable PR bitstream verification, the Compiler generates both a public Secure Mask Settings File (`.smsf`) and private Partially Masked SRAM Object file (`.pmsf`) for each PR region during the base compilation.

The `.pmsf` contains comprehensive information that the Programmer requires to generate the PR bitstream for a Client region, including the actual bit settings, a region mask, and all the auxiliary bit masks. The `.smsf` contains a region ownership mask and comprehensive information to detect a peek or poke attack by the PR region’s persona.

Thereafter, the Programmer requires both the private `.pmsf` and public `.smsf` to generate the PR bitstream for this PR region, ensuring that the PR persona can only change bits that the persona owns. The Platform Owner may or may not release `.smsf` files to third-party Clients as part of the PR region collateral. The Platform Owner uses the `.smsf` to generate the PR bitstream from Client’s `.pmsf` for this PR region with the Programmer.

Figure 38. PR Bitstream Security Validation in Programmer
Follow these steps to license, enable, and use PR bitstream security verification:

1. Obtain the license file to enable generation of .smsf files for PR regions during base compilation, and to perform PR bitstream security verification during PR bitstream generation in the Programmer. To obtain the license, login or register for a My-Intel account, and then submit an Intel Premier Support case requesting the license key.

2. To add the license file to the Intel Quartus Prime Pro Edition software, click **Tools ➤ License Setup** and specify the feature **License File**.

3. To enable PR security validation features, add the following line to the project .qsf:

   ```
   set_global_assignment -name PR_SECURITY_VALIDATION on
   ```

4. Compile the base revision.

5. Following base compilation, view the Assembler reports to view the generated .smsf files required for bitstream generation for each PR region.

6. The Platform Owner may release .smsf files to third-party clients as part of the PR region collateral. The Client provides the private .pmsf to the Platform Owner to verify PR security of the PR Persona configuration and generate validated PR bitstreams.

7. To validate PR security of the Client's .pmsf, the Platform Owner specifies the .smsf and corresponding .pmsf files at the Programmer command line to generate the validated PR bitstreams:

   ```
   quartus_cpf -c --smsf=<smsf_file> <pmsf_file> <output_file>
   ```

**Related Information**

My-Intel.com

### 1.12. PR Bitstream Compression and Encryption (Intel Arria 10 and Intel Cyclone 10 GX Designs)

You can compress and encrypt the base bitstream and the PR bitstream for your Intel Arria 10 and Intel Cyclone 10 GX PR project using options available in the Intel Quartus Prime software.

Compress the base and PR programming bitstreams independently, based on your design requirements. When encrypting only the base image, specify whether or not to encrypt the PR images. The following guidelines apply to PR bitstream compression and encryption:

- You can encrypt the base and PR image independently. You can use a non-volatile encryption key for the base image, and a volatile encryption key for the PR image.
- Refer to Table 9 on page 60 to ensure the correct Clock-to-Data (CD) ratio setting for encryption or compression.

Enable enhanced decompression by turning on the **Enable enhanced decompression** option when specifying the parameters in the IP Catalog or Platform Designer parameter editors.
1.12.1. Generating an Encrypted PR Bitstream (Intel Arria 10 or Intel Cyclone 10 GX Designs)

To partially reconfigure your Intel Arria 10 or Intel Cyclone 10 GX device with an encrypted bitstream:

**Note:** Intel Quartus Prime software does not support bitstream encryption and compression for Intel Stratix 10 or Intel Agilex designs.

1. Create a 256-bit key file (.key).
2. To generate the key programming file (.ekp) from the Intel Quartus Prime shell, type the following command:

```bash
quartus_cpf --key <keyfile>:<keyid> <base_sof_file> <output_ekp_file>
```

For example:

```bash
quartus_cpf --key my_key.key:key1 base.sof key.ekp
```

3. To generate the encrypted PR bitstream (.rbf), run the following command:

```bash
quartus_cpf -c <pr_pmsf_file> <pr_rbf_file>
qucrypt -e --keyfile=<keyfile> --keyname=<keyid> --lockto=<qlk file> --keystore=<battery|OTP> <pr_rbf_file> <pr_encrypted_rbf_file>
```

- `lockto`—specifies the encryption lock.
- `keystore`—specifies the volatile key (battery) or the non-volatile key (OTP).

For example:

```bash
quartus_cpf -c top_v1.pr_region.pmsf top_v1.pr_region.rbf 
qucrypt -e --keyfile=my_key.key --keyname=key1 --keystore=battery 
    --lockto=qlk_file top_v1.pr_region.rbf top_v1_encrypted.rbf
```

4. To program the key file as volatile key (default) into the device, type the following command:

```bash
quartus_pgm -m jtag -o P;<output_ekp_file>
```

For example:

```bash
quartus_pgm -m jtag -o P;key.ekp
```

5. To program the base image into the device, type the following command:

```bash
quartus_pgm -m jtag -o P;<base_sof_file>
```

For example:

```bash
quartus_pgm -m jtag -o P;base.sof
```
6. To partially reconfigure the device with the encrypted bitstream, type the following command:

\texttt{quartus_pgm -m jtag --pr <output\_encrypted\_rbf\_file>}

For example:

\texttt{quartus_pgm -m jtag --pr top\_v1\_encrypted.rbf}

\textit{Note:} \texttt{qcrypt} generates an error if the \textbf{Enable bitstream compatibility check} parameter is enabled for an instance of the Partial Reconfiguration Controller Intel Arria 10/ Cyclone 10 FPGA IP. Use one of the following methods to avoid this error:

- Use the \textbf{Convert Programming Files} dialog box, rather than \texttt{qcrypt}, to generate the encrypted PR bitstream, as \textit{Generating PR Bitstream Files} describes.

- If you want use \texttt{qcrypt} with Intel Arria 10 or Intel Cyclone 10 GX designs, regenerate the Partial Reconfiguration Controller IP without the \textbf{Enable bitstream compatibility check} option enabled, and with the \textbf{Enable hierarchical PR support} option enabled, as \textit{Adding the Partial Reconfiguration Controller Intel Arria 10/Cyclone 10 FPGA IP} describes. Recompile the design before regenerating the PR bitstream.

\textbf{Related Information}

- AN 556: Using the Design Security Features in Intel FPGAs
- \textit{Generating PR Bitstream Files}
- \textit{Adding the Partial Reconfiguration Controller Intel Arria 10/Cyclone 10 FPGA IP}

1.12.2. Clock-to-Data Ratio for Bitstream Encryption and Compression (Intel Arria 10 or Intel Cyclone 10 GX Designs)

The following table lists the valid combinations of bitstream encryption and compression. The Clock-to-Data (CD) ratio is defined as the number of clock cycles that each cycle of data must remain valid before the next clock cycle. For example, a CD ratio of 4 means that the data must remain valid for 4 clock cycles before the next cycle. Enhanced decompression uses the same CD ratio as plain bitstreams (that is, with both encryption and compression off). When enhanced compression is enabled, always refer to x16 data width. If you use compression and enhanced compression together, the CD ratio follows the compression bitstream - 4. If you use plain and enhanced compression together, the CD ratio follows the plain bitstream - 1.

\begin{table}[h]
\centering
\caption{Valid Combinations and CD Ratio for Bitstream Encryption and Compression}
\begin{tabular}{|c|c|c|c|}
\hline
Configuration Data Width & AES Encryption & Basic Compression & CD Ratio \\
\hline
\textbf{x8} & Off & Off & 1 \\
 & Off & On & 2 \\
 & On & Off & 1 \\
\hline
\textbf{x16} & Off & Off & 1 \\
 & Off & On & 4 \\
 & On & Off & 2 \\
\hline
\end{tabular}
\end{table}

\textit{continued...}
Use the exact CD ratio that the *Valid combinations and CD Ratio for Bitstream Encryption and Compression* table specifies for different bitstream types. The CD ratio for plain `.rbf` must be 1. The CD ratio for compressed `.rbf` must be 2, 4 or 8, depending on the width. Do not specify the CD ratio as the necessary minimum to support different bitstream types.

*Note:* Intel Quartus Prime software does not support bitstream encryption and compression for Intel Stratix 10 or Intel Agilex designs.

### 1.12.3. Data Compression Comparison

Standard compression results in a 30-45% decrease in `.rbf` size. Use of the enhanced data compression algorithm results in 55-75% decrease in `.rbf` size. The algorithm increases the compression at the expense of additional core area required to implement the compression algorithm.

The following figure shows the compression ratio comparison across PR designs with varying degrees of Logic Element (LE):

#### Figure 39. Compression Ratio Comparison between Standard Compression and Enhanced Compression

![Compression Ratio Comparison Figure](image)

### 1.13. Avoiding PR Programming Errors

You can use the following guidelines to avoid or resolve common PR programming errors.
<table>
<thead>
<tr>
<th>PR Programming Guideline</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device in project must match device on board</td>
<td>Confirm the target FPGA device that you specify for the project matches the device on the development kit you target. These two devices must be the same. Click Assignments ➤ Device to view the target device.</td>
</tr>
<tr>
<td>Programmer versions must match</td>
<td>When using the Intel Quartus Prime Programmer for PR programming, confirm that the Programmer version matches the Intel Quartus Prime version that you use for compilation. A mismatch between the Programmer and Intel Quartus Prime software version can occur if you compile on one machine, and then program on a different machine with a different Intel Quartus Prime version. The software version match is especially critical for Intel Stratix 10 and Intel Agilex designs because the PR configuration hardware has dependencies inside the Programmer.</td>
</tr>
</tbody>
</table>
| Specify a lower JTAG clock frequency                         | Lower the JTAG clock frequency to 6MHz:  
1. In the Programmer window, click Hardware Setup, and then select Intel FPGA Download Cable II as the programming hardware.  
2. For the Hardware frequency, specify a value from 24000000 (24MHz) to 6000000 (6MHz).                                                                                                                                                                                      |
| Close timing for all revisions                               | Confirm that each project revision closes timing after design compilation:  
1. In the Compilation Report, expand the Timing Analyzer ➤ Slow 900mV 100C Model folders, and then view the Setup Summary, Hold Summary, Recovery Summary, Removal Summary, and Minimum Pulse Width Summary reports. In each report, verify that there are no timing violations indicated by a negative Slack value in the report.  
2. Repeat step 1 to verify timing closure in the Slow 900mV 0C Model, the Fast 900mV 100C Model, and the Fast 900mV 0C Model. The design closes timing when there are no negative Slack values for any clock in the report.                                                                                     |
<table>
<thead>
<tr>
<th>PR Programming Guideline</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3. Repeat steps 1 and 2 for each project revision in the PR design.</td>
<td></td>
</tr>
</tbody>
</table>

**Note:** If an error occurs during PR operation for Intel Stratix 10 or Intel Agilex designs using Single Event Upset (SEU) detection, the PR region is frozen, becomes non-functional, and SEU disables for all sectors the PR region covers. The Avalon streaming status interface or the Avalon memory-mapped register map of the Partial Reconfiguration Controller Intel FPGA IP reflects this error status. To resolve this error and restore the SEU detection, perform another PR operation to reload a valid PR bitstream.

### 1.14. Exporting a Version-Compatible Compilation Database for PR Designs

Using version-compatible databases, you can import the base revision of a PR design to a later version of the Intel Quartus Prime software, and then compile the PR revisions in the later version of software, without recompiling the static region.

This technique is helpful when you want to compile and generate bitstreams for the PR implementation revisions with a later version of the Intel Quartus Prime software. Configuration bitstreams are not version compatible, and you must generate all bitstreams from the same version of Intel Quartus Prime software.

After migrating the base revision to a later version of the Intel Quartus Prime software, the bitstream you generate is only compatible with bitstreams from PR implementation compilations using that same Intel Quartus Prime software version. Such a bitstream is incompatible with the PR bitstreams from an earlier version of the Intel Quartus Prime software.
The Intel Quartus Prime Pro Edition software version supports version-compatible databases for PR designs for the following software versions and devices:

### Table 11. Version-Compatible Compilation Database Device and Software Version Support for PR Designs

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>18.0</td>
<td>19.1</td>
<td>No Support.</td>
<td>Supports all devices.</td>
</tr>
<tr>
<td>18.1</td>
<td>19.1</td>
<td>• 1SG250L</td>
<td>Supports all devices.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1SG280H_S2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1SG280L</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1SG280L_S3</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1SX250L</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1SX280L</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1SX280L_S3</td>
<td></td>
</tr>
<tr>
<td>19.1</td>
<td>19.3</td>
<td>• 1SM16BH</td>
<td>Supports all devices.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1SM21BH</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1SM16CH</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1SM21CH</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1SM21KH</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1SM16KH</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1SM21LH</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1SM16LH</td>
<td></td>
</tr>
<tr>
<td>19.3</td>
<td>20.1</td>
<td>• 1SG10MH_U1</td>
<td>Supports all devices.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1SG10MH_U2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1ST250E</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1ST280E</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1SM16E</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1SM21E</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1ST165E</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1ST210E</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1SG166H</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1SG211H</td>
<td></td>
</tr>
<tr>
<td>20.1</td>
<td>20.3</td>
<td>• 1SD280P</td>
<td>Supports all devices.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1ST040E</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1ST085E</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1ST110E</td>
<td></td>
</tr>
</tbody>
</table>

The following topics describe the version-compatible database generation flow and steps.

- **Version-Compatible Database Flow for PR Designs** on page 65
- **Generating a Version-Compatible Compilation Database for PR Designs** on page 65

**Related Information**

  - For more information on version-compatible compilation database file generation.

Migrating a design with a single PR region involves the following high-level steps:

1. Perform initial compilation of the base revision in the Intel Quartus Prime software version $N$.
2. Export a version-compatible database for the entire design in the Intel Quartus Prime software version $N$.
3. Import the version-compatible database into the Intel Quartus Prime software version $N+M$.
4. Generate the base revision .sof file and bitstreams with the Intel Quartus Prime Assembler version $N+M$.
5. Export the static region .qdb in the Intel Quartus Prime software version $N+M$.
6. Perform a PR implementation compile in the Intel Quartus Prime software version $N+M$.

**Note:** You must generate all of the PR bitstreams that you use with the Intel Quartus Prime software version ($N+M$), including the full-chip configuration bitstream and the PR bitstream .rbf.

![Figure 40. Static Region Migration (Single PR Region Compiled in Later Version)](image)

1.14.2. Generating a Version-Compatible Compilation Database for PR Designs

Follow these steps to generate a version-compatible compilation database for PR designs:

1. Export the entire compiled design from the Intel Quartus Prime software version $N$ by clicking **Project ➤ Export Design**, or by command line:

```
quartus_cdb <project> -c <base_revision> --export_design --snapshot_final --file <base_revision>.qdb
```

2. Import the compiled design to the Intel Quartus Prime software version $N+M$ by clicking **Project ➤ Import Design**, or by command line:

```
quartus_cdb <project> -c <base_revision_import> --import_design --file <base_revision>.qdb
```
Note: Whenever possible, import the design into a different working directory than the directory that you use to compile the base design. If you must use the same directory for import and for compiling the base design, make a backup copy of your compiled design by archiving that design with qdb/* included, or make a copy of the entire directory and subdirectories elsewhere. You must also remove the old database directory qdb/* and all the bitstream related files (*.sof, *.msf, *.pmsf).

3. Rerun the finalize stage of the Fitter in the Intel Quartus Prime Pro Edition software version \( N+M \) by clicking Processing ➤ Start ➤ Start Fitter (Finalize), or by command line:

```
quartus_fit <project> -c <base_revision_import> --finalize
```

4. Run the Assembler in the Intel Quartus Prime Pro Edition software version \( N+M \) to regenerate the static region bitstream by clicking Processing ➤ Start ➤ Start Assembler, or by command line:

```
quartus_asm <project> -c <base_revision_import>
```

5. Export the static region .qdb in the Intel Quartus Prime Pro Edition software version \( N+M \) by clicking Project ➤ Export Design Partition, or by command line:

```
quartus_cdb <project> -c <base_revision_import> --export_block \nroot_partition --snapshot final --file --include_sdc_entity_in_partition static.qdb
```

Note: When exporting the base revision and the static partition, you must include any .sdc files that apply to the partition, by using the include_sdc_entity_in_partition option.

6. Compile each implementation revision in Intel Quartus Prime Pro Edition software version \( N+M \), using the static revision .qdb that you exported in the previous step.

```
quartus_sh -flow compile <project> -c <impl_rev>
```

### 1.15. Creating a Partial Reconfiguration Design Revision History

<table>
<thead>
<tr>
<th>Document Version</th>
<th>Intel Quartus Prime Version</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>2020.12.11</td>
<td>20.3</td>
<td>• Corrected typo in PMSF file definition in &quot;Using PR Bitstream Security Verification&quot;</td>
</tr>
</tbody>
</table>
| 2020.09.28        | 20.3                        | • Added note about preserving SDC files to "Step 8: Setup PR Implementation Revisions" topic.  
• Added new "Using Parent QDB Files from Different Compiles" topic.  
• Added "PR Design Timing Closure Best Practices" topic.  
• Replaced references to Avalon-MM and Avalon-ST with Avalon memory-mapped and Avalon streaming for legal compliance. |

continued...
<table>
<thead>
<tr>
<th>Document Version</th>
<th>Intel Quartus Prime Version</th>
<th>Changes</th>
</tr>
</thead>
</table>
| 2020.08.07       | 20.2                        | • Added screenshot and details about Synthesis report to "Evaluating PR Region Initial Conditions" topic.  
• Added details about Include entity-bound SDC files option requirements to "Step 7: Compile the Base Revision and Export the Static Region" topic.  
• Removed stated support for "PR Bitstream Security Verification" for Intel Agilex devices. This feature is not yet supported for Intel Agilex devices.  
• Added details about Include entity-bound SDC files option requirements to "Generating a Version-Compatible Compilation Database" for PR Designs topic.  
| 2020.06.22       | 20.2                        | • Added PR simulation support for Intel Agilex designs. |
| 2020.05.11       | 20.1                        | • Revised description of PR bitstream compatibility checking steps for Intel Cyclone 10 GX and Intel Arria 10 devices. |
| 2020.04.13       | 20.1                        | • Updated requirements in "Partial Reconfiguration Bitstream Compatibility Checking" topic.  
• Added note about time borrowing to "Partial Reconfiguration Design Timing Analysis" topic.  
• Added note indicating HPR designs do not support PR security bitstream verification. |
| 2019.11.18       | 19.3.0                      | • Changed title of "Migrating PR Regions to a Later Software Version" to "Exporting a Version-Compatible Compilation Database for a PR Design," generalized examples, and removed INI requirement. |
| 2019.09.30       | 19.3.0                      | • Added compilation support for Intel Cyclone 10 GX and Intel Agilex PR designs.  
• Updated IP name from "Partial Reconfiguration Controller Intel Stratix 10 FPGA IP" to "Partial Reconfiguration Controller Intel FPGA IP" to encompass Intel Agilex designs.  
• Updated wording of "Clock Gating" topic for clarity.  
• Added note to Partial Reconfiguration Controller Intel Arria 10/Cyclone 10 FPGA IP "Parameters" topic about support for enhanced decompression. |
| 2019.06.10       | 19.1.0                      | • Added details about synthesis of PRESERVE_FANOUT_FREE_NODE to "Partial Reconfiguration Design Guidelines." |
| 2019.04.22       | 19.1.0                      | • Indicated support for POF generation support for Intel Cyclone GX devices.  
• Corrected code example in "PR Migration Flow" topic. |
| 2019.04.01       | 19.1.0                      | • Described migration of the static region of a PR design to a later version of the Intel Quartus Prime software.  
• Described new "PR Bitstream Security Validation" feature.  
• Described new location of auto export from output_files to project directory. |
| 2018.12.30       | 18.1.1                      | • Described "Partial Reconfiguration Bitstream Compatibility Check" and PR region limitations. |
| 2018.10.24       | 18.1.0                      | • Added "PR File Management" topic.  
• Updated first guideline in "Partial Reconfiguration Design Guidelines." |

continued...
<table>
<thead>
<tr>
<th>Document Version</th>
<th>Intel Quartus Prime Version</th>
<th>Changes</th>
</tr>
</thead>
</table>
| 2018.09.24       | 18.1.0                      | - Described automated .qdb partition export in "Exporting a Design Partition."
|                  |                             | - Added details about required assignments to "Step 6: Create Revisions for Personas."
|                  |                             | - Removed references to placed snapshot. Only synthesized and final snapshots are supported.
|                  |                             | - Corrected description of Entity Re-binding option in Design Partition Settings table.
|                  |                             | - Added command line instructions for creating a revision.
|                  |                             | - Stated PR compilation flow support for Intel Cyclone 10 GX devices.
|                  |                             | - Updated Partial Reconfiguration Controller Intel Arria 10 FPGA IP name to Partial Reconfiguration Controller Intel Arria 10/Cyclone 10 FPGA IP.
|                  |                             | - Added "Viewing Row Clock Region Boundaries."
|                  |                             | - Added "Planning Clocks and other Global Routing."
| 2018.07.18       | 18.0.0                      | - Corrected signals in Simulation of PR Persona Switching diagram. |
| 2018.06.18       | 18.0.0                      | - Corrected syntax errors and added note in Running Timing Analysis on Aggregate Revisions. |
| 2018.05.29       | 18.0.0                      | - Added description of "/" that identifies the root partition hierarchy path in Design Partitions Window. 
|                  |                             | - Clarified .qsf assignment in Running Timing Analysis on Aggregate Revisions. |
| 2018.05.07       | 18.0.0                      | - Added description of new Partial Reconfiguration External Configuration Controller Intel Stratix 10 FPGA IP. 
|                  |                             | - Removed descriptions of obsolete synthesis-only revisions and corresponding personas. Replaced with latest simplified flow instructions. 
|                  |                             | - Updated names of Partial Reconfiguration Controller Intel Arria 10 FPGA IP and Partial Reconfiguration Controller Intel Stratix 10 FPGA IP. 
|                  |                             | - Added Design Partition Settings topic. 
|                  |                             | - Added Evaluating PR Partition Initial Conditions topic. 
|                  |                             | - Added Avoiding PR Programming Errors topic. 
|                  |                             | - Described qcrypt incompatibility with Enable bitstream compatibility check and workaround. 
|                  |                             | - Added as chapter in new Partial Reconfiguration User Guide. 
|                  |                             | - Updated command-line syntax in Running Timing Analysis on Aggregate Revisions topic. 
|                  |                             | - Removed obsolete HPR flow script information and linked to AN826: Hierarchical Partial Reconfiguration Tutorial for Intel Stratix 10 GX FPGA Development Board 
|                  |                             | - Added note about recovery after PR error when using SEU detection in Intel Stratix 10 designs. |
| 2017.11.06       | 17.1.0                      | - Added partial reconfiguration support for Intel Stratix 10 devices. 
|                  |                             | - Added descriptions of Intel Stratix 10 Partial Reconfiguration Controller IP, SUPR, HPR, and SDM to terms list. 
|                  |                             | - Updated for latest Intel branding and software user interface. |
| 2017.05.08       | 17.0.0                      | - Added information about Hierarchical Partial Reconfiguration. 
|                  |                             | - Added new topic Partial Reconfiguration Simulation and Verification. 
|                  |                             | - Added new topic 'Run Timing Analysis on a Design with Multiple PR Partitions'. 
|                  |                             | - Updated Freeze Logic for PR Regions. 
|                  |                             | - Added new topic Debugging Using Signal Tap Logic Analyzer. 
|                  |                             | - Other minor updates. |
2. Partial Reconfiguration Solutions IP User Guide

The Intel Quartus Prime Pro Edition software includes the following Intel FPGA IP cores that simplify partial reconfiguration implementation.

Instantiate one or more of these IP cores to implement handshake and freeze logic for PR functionality in your design. Alternatively, create your own PR handshake and freeze logic that interfaces with the PR region.

Table 12. Partial Reconfiguration IP Cores

<table>
<thead>
<tr>
<th>Intel FPGA IP</th>
<th>Description</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partial Reconfiguration Controller Intel FPGA IP</td>
<td>Dedicated IP component that sends the partial reconfiguration bitstream for the Intel Stratix 10 or Intel Agilex FPGAs. The PR bitstream performs reconfiguration by adjusting CRAM bits in the FPGA.</td>
<td>One instance per Intel Stratix 10 or Intel Agilex FPGA</td>
</tr>
<tr>
<td>Partial Reconfiguration External Configuration Controller Intel FPGA IP</td>
<td>IP component that supports Intel Stratix 10 and Intel Agilex FPGA partial reconfiguration via an external source over dedicated PR pins.</td>
<td>One instance per Intel Stratix 10 or Intel Agilex FPGA for external configuration</td>
</tr>
<tr>
<td>Partial Reconfiguration Controller Intel Arria 10/Cyclone 10 FPGA IP</td>
<td>Dedicated IP component that sends the partial reconfiguration bitstream to the Intel Arria 10 or Intel Cyclone 10 GX FPGA. The PR bitstream performs reconfiguration by adjusting CRAM bits in the FPGA.</td>
<td>One instance per Intel Arria 10 or Intel Cyclone 10 GX FPGA, internal or external configuration.</td>
</tr>
<tr>
<td>Partial Reconfiguration Region Controller Intel FPGA IP</td>
<td>Provides a standard Avalon memory-mapped interface to the block that controls handshaking with the PR region. Ensures that PR region stops, resets, and restarts, according to the PR handshake.</td>
<td>One instance per PR region.</td>
</tr>
<tr>
<td>Avalon Memory-Mapped Partial Reconfiguration Freeze Bridge Intel FPGA IP</td>
<td>Provides freeze capabilities to the PR region for Avalon memory-mapped interfaces.</td>
<td>One instance for each interface in each PR region.</td>
</tr>
<tr>
<td>Avalon Streaming Partial Reconfiguration Freeze Bridge Intel FPGA IP</td>
<td>Provides freeze capabilities to the PR region for Avalon streaming interfaces.</td>
<td>One instance for each interface in each PR region.</td>
</tr>
</tbody>
</table>

2.1. Internal and External PR Host Configurations

You perform PR with either an internal host residing in the core resources, or with an external host via dedicated device pins. Use of an internal host stores all PR host logic on the FPGA device, rather than on an external device. The PR host interfaces with the control block through simple handshaking and data transfer.

(5) The Intel Quartus Prime Pro Edition software supports compilation of PR designs for Intel Agilex devices, but does not yet support Intel Agilex PR bitstream generation.
Figure 41. Intel Arria 10 or Intel Cyclone 10 GX Partial Reconfiguration IP Components (Internal Host)

Figure 42. Intel Stratix 10 or Intel Agilex Partial Reconfiguration IP Components (Internal Host)
Figure 43. Partial Reconfiguration with Microcontroller External Host (Intel Arria 10 or Intel Cyclone 10 GX device)

Microcontroller (Host)

PR Region Controller IP
Freeze Control Block
Conduit Splitter
Conduit Merger
Static Region

Arria 10 PR Controller IP
MM Slave
PR Control Block (hard)

Intel Arria 10 FPGA

Figure 44. Partial Reconfiguration with HPS Internal Host (Intel Arria 10 or Intel Cyclone 10 GX device)

HPS (Host)

FPGA Manager
H2F AXI Master

PR Region Controller IP
Freeze Control and Status Register
MM Slave

Intel Arria 10 FPGA

Illegal request
Illegal request [n:0]
Reset_n
Clk

Start_req
Start_ack
Stop_req
Stop_ack
Region_reset
Region_reset

Partial Reconfiguration Solutions IP User Guide
UG-20136 | 2020.12.11
2.2. Partial Reconfiguration Controller Intel FPGA IP

The Partial Reconfiguration Controller Intel FPGA IP provides partial reconfiguration functionality for Intel Stratix 10 and Intel Agilex designs. The IP core provides a standard interface to the FPGA secure device manager (SDM), and has a maximum clock frequency of 200 MHz.\(^6\)

Figure 45. Intel Stratix 10 and Intel Agilex Partial Reconfiguration Controller (Avalon Streaming Interface)

\(^7\)

Note: If an error occurs during PR operation for Intel Stratix 10 or Intel Agilex designs using Single Event Upset (SEU) detection, the PR region is frozen, becomes non-functional, and SEU disables for all sectors the PR region covers. The Avalon streaming status interface or the Avalon memory-mapped register map of the Partial Reconfiguration Controller Intel FPGA IP reflects this error status. To resolve this error and restore the SEU detection, perform another PR operation to reload a valid PR bitstream.

2.2.1. Memory Map

The Partial Reconfiguration Controller Intel FPGA IP has the following memory map.

Table 13. Avalon Memory-Mapped Slave Memory Map

<table>
<thead>
<tr>
<th>Name</th>
<th>Address Offset</th>
<th>Width</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PR_DATA</td>
<td>0x00</td>
<td>32</td>
<td>Write</td>
<td>Every data write to this address indicates this bitstream is sending to the IP core.</td>
</tr>
</tbody>
</table>

\(^6\) The Intel Quartus Prime Pro Edition software supports compilation of PR designs for Intel Agilex devices, but does not yet support Intel Agilex PR bitstream generation.

\(^7\) Avalon memory mapped interface variant also available.
Width is set by the *Input data width* parameter.

### PR_CSR

<table>
<thead>
<tr>
<th>Name</th>
<th>Address Offset</th>
<th>Width</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PR_CSR</td>
<td>0x01</td>
<td>32</td>
<td>Read or Write</td>
<td>Control and status registers with the following offset bits:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• 31 - 6: Reserved.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• 5: Read/Write for irq signal mask bit. Write 1 to this bit enable irq signal and 0 to disable the irq signal.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• 4: Read/Clear for irq signal. The irq signal asserts if an error occurs. The Master must read the status signal and clear the interrupt by writing 1 to this bit.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• 3 - 1: Read-only for status signal.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• 0: Read/Write for pr_start signal. To streamline the flow, the IP core automatically de-asserts to value 0, one clock cycle after the signal asserts.</td>
</tr>
</tbody>
</table>

### PR_SW_VER

<table>
<thead>
<tr>
<th>Name</th>
<th>Address Offset</th>
<th>Width</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PR_SW_VER</td>
<td>0x02</td>
<td>32</td>
<td>Read</td>
<td>Read-only SW version register. Register is currently 0xBA500000.</td>
</tr>
</tbody>
</table>

**Note:** For IP core instantiation guidelines, refer to the appropriate device configuration user guide.

**Related Information**
- Avalon Interface Specifications
- Clock Networks and PLLs in Arria 10 Devices
- Intel Stratix 10 Configuration User Guide
- Intel Agilex Configuration User Guide

### 2.2.2. Parameters

The Partial Reconfiguration Controller Intel FPGA IP supports customization of the following parameters.

**Table 14. Partial Reconfiguration Controller Intel FPGA IP Parameter Settings**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable Avalon-ST sink or Avalon-MM slave interface</td>
<td>Avalon-ST/ Avalon-MM</td>
<td>Enables the controller’s Avalon streaming sink or Avalon memory-mapped slave interface.</td>
</tr>
<tr>
<td>Input data width</td>
<td>&lt;bits&gt;</td>
<td>Specifies the size of the controller’s data conduit interface in bits. The IP supports device widths of 32 and 64.</td>
</tr>
<tr>
<td>Enable interrupt interface</td>
<td>Yes/No</td>
<td>Enables interrupt assertion for detection of incompatible bitstream, CRC_ERROR, PR_ERROR, or successful partial reconfiguration. Upon interrupt, query PR_CSR[4:2] for status. Write a 1 to PR_CSR[5] to clear the interrupt. Use only together with the Avalon memory-mapped slave interface.</td>
</tr>
<tr>
<td>Enable protocol error</td>
<td></td>
<td>Reads out the error bit from the CSR register.</td>
</tr>
</tbody>
</table>
2.2.3. Ports

The Partial Reconfiguration Controller Intel FPGA IP includes the following interface ports.

Figure 47.  Avalon Streaming Sink Interface Ports
Figure 48. Avalon Memory-Mapped Slave Interface Ports

Table 15. Clock/Reset Ports

<table>
<thead>
<tr>
<th>Port Name</th>
<th>Width</th>
<th>Direction</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>reset</td>
<td>1</td>
<td>Input</td>
<td>Asynchronous reset for the PR Controller IP core. Resetting the PR Controller IP core during a partial reconfiguration operation can cause the device to lock up.</td>
</tr>
<tr>
<td>clk</td>
<td>1</td>
<td>Input</td>
<td>Input clock to the PR Controller IP core. The input clock must be free-running. The IP core has a maximum clock frequency of 200 MHz.</td>
</tr>
</tbody>
</table>

Table 16. Avalon Streaming Slave Interface Ports

These ports are available when you enable the Avalon Streaming slave interface.

<table>
<thead>
<tr>
<th>Port Name</th>
<th>Width</th>
<th>Direction</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>pr_start</td>
<td>1</td>
<td>Input</td>
<td>A signal arriving at this port asserted high initiates a PR event. You must assert this signal high for a minimum of one clock cycle, and de-assert it low, prior to the end of the PR operation.</td>
</tr>
<tr>
<td>avst_sink_data</td>
<td>32</td>
<td>64</td>
<td>Input</td>
</tr>
<tr>
<td>avst_sink_valid</td>
<td>1</td>
<td>Input</td>
<td>Avalon streaming data valid signal that indicates the avst_sink_data port contains valid data.</td>
</tr>
<tr>
<td>avst_sink_ready</td>
<td>1</td>
<td>Output</td>
<td>Avalon streaming ready signal that indicates the device is ready to read the streaming data on the avst_sink_data port whenever the avst_sink_valid signal asserts high. Stop sending valid data when this port is low.</td>
</tr>
<tr>
<td>status[2..0]</td>
<td>3</td>
<td>Output</td>
<td>A 3-bit error output that indicates the status of a PR event. Once the outputs latch high as follow, you can only reset the outputs at the beginning of the next PR event: 3'b000 – power-up nreset asserted 3'b001 – configuration system is busy 3'b010 – PR operation is in progress 3'b011 – PR operation successful 3'b100 – PR_ERROR is triggered 3'b101 – Reserved</td>
</tr>
</tbody>
</table>

continued...
<table>
<thead>
<tr>
<th>Port Name</th>
<th>Width</th>
<th>Direction</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>protocol_error</td>
<td>1</td>
<td>Output</td>
<td>Reads out the error bit from the CSR register.</td>
</tr>
<tr>
<td>avmm_slave_address</td>
<td>4</td>
<td>Input</td>
<td>Avalon memory-mapped address bus in the unit of Word addressing.</td>
</tr>
<tr>
<td>avmm_slave_read</td>
<td>1</td>
<td>Input</td>
<td>Avalon memory-mapped read control.</td>
</tr>
<tr>
<td>avmm_slave_readdata</td>
<td>32</td>
<td>Output</td>
<td>Avalon memory-mapped read data bus.</td>
</tr>
<tr>
<td>avmm_slave_write</td>
<td>1</td>
<td>Input</td>
<td>Avalon memory-mapped write control.</td>
</tr>
<tr>
<td>avmm_slave_writedata</td>
<td>32</td>
<td>Input</td>
<td>Avalon memory-mapped write data bus.</td>
</tr>
<tr>
<td>avmm_slave_waitrequest</td>
<td>1</td>
<td>Output</td>
<td>Upon assertion, indicates that the IP is busy and the IP is unable to respond to a read or write request.</td>
</tr>
<tr>
<td>irq</td>
<td>1</td>
<td>Output</td>
<td>Interrupt signal when you enable the Enable interrupt interface parameter.</td>
</tr>
</tbody>
</table>
2.3. Partial Reconfiguration Controller Intel Arria 10/Cyclone 10 FPGA IP

The Partial Reconfiguration Controller Intel Arria 10/Cyclone 10 FPGA IP provides a standard interface to the partial reconfiguration functionality in the PR control block. Use this IP core to avoid manually instantiating a PR control block interface. The Partial Reconfiguration Controller Intel Arria 10/Cyclone 10 FPGA IP supports Intel Arria 10 and Intel Cyclone 10 GX PR designs with a maximum clock frequency of 100MHz.

Figure 50. Partial Reconfiguration Controller Intel Arria 10/Cyclone 10 FPGA IP

2.3.1. Slave Interface

The Partial Reconfiguration Controller Intel Arria 10/Cyclone 10 FPGA IP provides an Avalon memory-mapped slave interface to read and write to PR configuration registers.

Table 18. Data/CSR Memory Map Format

<table>
<thead>
<tr>
<th>Name</th>
<th>Address Offset</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PR_DATA</td>
<td>0x00</td>
<td>Write</td>
<td>Every data write to this address indicates this bitstream is sent to the IP core. Performing a read on this address returns all 0’s.</td>
</tr>
<tr>
<td>PR_CSR</td>
<td>0x01</td>
<td>Read or Write</td>
<td>Control and status registers.</td>
</tr>
<tr>
<td>Version Register</td>
<td>0x02</td>
<td>Read-Only</td>
<td>Read-only SW version register. Register is currently 0xAA500003</td>
</tr>
<tr>
<td>PR Bitstream ID</td>
<td>0x03</td>
<td>Read-Only</td>
<td>Read-only PR POF ID register</td>
</tr>
</tbody>
</table>
Table 19.  PR_CSR Control and Status Registers

<table>
<thead>
<tr>
<th>Bit Offset</th>
<th>Description</th>
</tr>
</thead>
</table>
| 0          | Read and write control register for pr_start signal. Refer to Ports on page 82 for details on the pr_start signal.  
pr_start = PR_CSR[0]  
The IP core deasserts PR_CSR[0] to value 0 automatically, one clock cycle after the PR_CSR[0] asserts. This streamlines the flow to avoid manual assertion and de-assertion of this register to control pr_start signal. |
| 1          | Reserved.                                                                   |
| 2-4        | Read-only status register for status[2:0] signal.  
PR_CSR[4:2] = status[2:0]  
Refer to Ports on page 82 for details on the status signals. |
| 5          | Read and clear bit for interrupt. If you enable the interrupt interface, reading this bit returns the value of the irq signal. Writing a 1 clears the interrupt.  
If you disable the interrupt interface, reading this bit always returns a value of 0. |
| 0-31       | Reserved bits. Depends on the Avalon memory-mapped data bus width. |

Related Information
- Avalon Interface Specifications
- Clock Networks and PLLs in Arria 10 Devices

2.3.2. Reconfiguration Sequence

Partial reconfiguration occurs through the Avalon memory-mapped slave interface in the following sequence:

1. Avalon memory-mapped master component writes 0x01 to IP address offset 0x1 to trigger PR operation.
2. Optionally poll the status register until PR Operation in Progress. Not polling results in waitrequest on first word.
3. Avalon memory-mapped master component writes PR bitstream to IP address offset 0x0, until all the PR bitstream writes. When enhanced decompression is on, waitrequest activates throughout the PR operation. Ensure that your master can handle waitrequest from the slave interface.
4. Avalon memory-mapped master component reads the data from IP address offset 0x1 to check the status[2:0] value. Optionally, the Avalon memory-mapped master component reads the status[2:0] of this IP during a PR operation to detect any early failure, for example, PR_ERROR.

2.3.3. Interrupt Interface

If you enable the Avalon Memory Mapped Slave interface, you can use the optional interrupt interface of the Partial Reconfiguration Controller Intel Arria 10/Cyclone 10 FPGA IP.
The IP core asserts irq during the following events:

**Table 20. Interrupt Interface Events**

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>3'b001</td>
<td>PR_ERROR occurred.</td>
</tr>
<tr>
<td>3'b010</td>
<td>CRC_ERROR occurred.</td>
</tr>
<tr>
<td>3'b011</td>
<td>The IP core detects an incompatible bitstream.</td>
</tr>
<tr>
<td>3'b101</td>
<td>The result of a successful PR operation.</td>
</tr>
</tbody>
</table>

After irq asserts, the master performs one or more of the following:

- Query for the status of the PR IP core; PR_CSR[4:2].
- Carry out some action, such as error reporting.
- Once the interrupt is serviced, clear the interrupt by writing a "1" to PR_CSR[5].
The Partial Reconfiguration Controller Intel Arria 10/Cyclone 10 FPGA IP supports customization of the following parameters.

### Table 21. Parameter Settings

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use as partial reconfiguration internal host</td>
<td>On/Off</td>
<td>Enables the controller for use as an internal host. Enabling this option auto-instantiates prblock and crcblock WYSIWYG as part of your design. Disable this option to use the controller as an external host. Connect additional interface signals to the dedicated partial reconfiguration pins.</td>
</tr>
<tr>
<td>Enable JTAG debug mode</td>
<td>On/Off</td>
<td>Enables access to the controller by the Intel Quartus Prime Programmer for partial reconfiguration over a JTAG interface.</td>
</tr>
<tr>
<td>Enable Avalon-MM slave interface</td>
<td>On/Off</td>
<td>Enables the controller's Avalon memory-mapped slave interface. When this setting is Off, the IP controller enables the conduit interface.</td>
</tr>
<tr>
<td>Enable freeze interface</td>
<td>On/Off</td>
<td>Enables the controller's single-bit freeze interface. This interface identifies whether any region in the design is active or frozen for partial reconfiguration operations. Leave this interface off, and use the freeze interface from the Partial Reconfiguration Region Controller IP.</td>
</tr>
<tr>
<td>Enable bitstream compatibility check</td>
<td>On/Off</td>
<td>Enables bitstream compatibility checks during partial reconfiguration operation from the external host. Bitstream compatibility check automatically enables when you use partial reconfiguration by internal host. Specify the partial reconfiguration bitstream ID value if you enable this option for partial reconfiguration by external host.</td>
</tr>
<tr>
<td>PR bitstream ID</td>
<td>&lt;32-bit integer&gt;</td>
<td>Specifies a signed, 32-bit integer value of the partial reconfiguration bitstream ID for the external host. This value must match the partial reconfiguration bitstream ID that the Compiler generates for the target partial reconfiguration design. Locate the partial reconfiguration bitstream ID of the target partial reconfiguration design in the Assembler report (.asm.rpt).</td>
</tr>
<tr>
<td>Input data width</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>Clock-to-data ratio</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>Divide error detection frequency by</td>
<td>1..256</td>
<td>Specifies the divide value of the internal clock. This value determines the frequency of the error detection CRC. The divide value must be a power of two. Refer to device documentation to determine the frequency of the internal clock for the device you select. Refer to Error Detection CRC Requirements on page 81.</td>
</tr>
<tr>
<td>Enable enhanced decompression</td>
<td>On/Off</td>
<td>Enable enhanced decompression of partial reconfiguration bitstreams. Note: You cannot use enhanced decompression together with encryption simultaneously. Enhanced decompression is only available with the Partial Reconfiguration Controller Intel Arria 10/Cyclone 10 FPGA IP.</td>
</tr>
</tbody>
</table>
Table 22. Advanced Settings

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Auto-instantiate partial reconfiguration control block</td>
<td>On</td>
<td>Off</td>
</tr>
<tr>
<td>Auto-instantiate CRC block</td>
<td>On</td>
<td>Off</td>
</tr>
<tr>
<td>Generate timing constraints file</td>
<td>On</td>
<td>Off</td>
</tr>
</tbody>
</table>

Figure 51. Partial Reconfiguration Controller Intel Arria 10/Cyclone 10 FPGA IP Parameter Editor

2.3.4.1. Error Detection CRC Requirements

The following describes requirements for enabling the error detection CRC option with various PR configuration method and parameter combinations. Click Assignments ➤ Device ➤ Device & Pin Options ➤ Error Detection CRC ➤ Enable Error Detection Check to enable EDCRC prior to PR bitstream generation.
Note: When using the Partial Reconfiguration Controller Intel Arria 10/Cyclone 10 FPGA IP with a 32-bit \textit{Input data width}, and with Passive Parallel \(x_1, x_8, \text{or } x_{16}\) configuration, you must turn on the \textbf{Enable Error Detection Check} option, and specify a \textbf{Divide error detection frequency by} value of 2 or 4.

Note: When using the Partial Reconfiguration Controller Intel Arria 10/Cyclone 10 FPGA IP with a 32-bit \textit{Input data width} and Passive Parallel \(x_{32}\) configuration, \(PR\) supports \textbf{Enable Error Detection Check} on or off. If \textbf{Enable Error Detection Check} is on, \(PR\) supports all values for \textbf{Divide error detection frequency by}.

Note: When using the Partial Reconfiguration Controller Intel Arria 10/Cyclone 10 FPGA IP with \(1, 8, \text{or } 16\)-bit \textit{Input data width}, and with Passive Parallel \(x_1, x_8, x_{16,} \text{or } x_{32}\) configuration, \(PR\) supports \textbf{Enable Error Detection Check} turned on or off. If \textbf{Enable Error Detection Check} is on, \(PR\) supports all values for \textbf{Divide error detection frequency by}.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|}
\hline
PR IP Input Data Width & Configuration Mode & Enable Error Detection Check & PR Support \\
\hline
1, 8, 16 & Passive Parallel \(x_1, x_8, x_{16}\) & Off & Yes \\
\hline
1, 8, 16 & Passive Parallel \(x_1, x_8, x_{16}\) & On & Yes, for all \textit{Divide error detection frequency by} values \\
\hline
32 & Passive Parallel \(x_1, x_8, x_{16}\) & Off & No support \\
\hline
32 & Passive Parallel \(x_1, x_8, x_{16}\) & On & Yes, for only \textit{Divide error detection frequency by} value 2 or 4 \\
\hline
1, 8, 16, 32 & Passive Parallel \(x_{32}\) & Off & Yes \\
\hline
1, 8, 16, 32 & Passive Parallel \(x_{32}\) & On & Yes, for all \textit{Divide error detection frequency by} values \\
\hline
\end{tabular}
\caption{Partial Reconfiguration Controller Intel Arria 10/Cyclone 10 FPGA IP Error Detection CRC (EDCRC) Requirements Summary}
\end{table}

2.3.5. Ports

The Partial Reconfiguration Controller Intel Arria 10/Cyclone 10 FPGA IP includes the following interface ports.
Figure 52. Partial Reconfiguration Controller Interface Ports (Internal Host)

Figure 53. Partial Reconfiguration Controller Interface Ports (External Host)
### Table 24. Clock/Reset Ports

<table>
<thead>
<tr>
<th>Port Name</th>
<th>Width</th>
<th>Direction</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>nreset</td>
<td>1</td>
<td>Input</td>
<td>Asynchronous reset for the PR Controller IP core. Resetting the PR Controller IP core during a partial reconfiguration operation initiates the withdrawal sequence.</td>
</tr>
<tr>
<td>clk</td>
<td>1</td>
<td>Input</td>
<td>User input clock to the PR Controller IP core. The IP core has a maximum clock frequency of 100MHz. The IP core ignores this signal during JTAG debug operations.</td>
</tr>
</tbody>
</table>

### Table 25. Freeze Interface Port

<table>
<thead>
<tr>
<th>Port Name</th>
<th>Width</th>
<th>Direction</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>freeze</td>
<td>1</td>
<td>Output</td>
<td>Active high signal that freezes the PR interface signals of any region undergoing partial reconfiguration. De-assertion of this signal indicates the end of PR operation. Use the Partial Reconfiguration Region Controller IP rather than the Partial Reconfiguration Controller IP freeze signal.</td>
</tr>
</tbody>
</table>

### Table 26. Conduit Interface Ports

<table>
<thead>
<tr>
<th>Port Name</th>
<th>Width</th>
<th>Direction</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>pr_start</td>
<td>1</td>
<td>Input</td>
<td>A 0 to 1 transition on this port initiates a PR event. You must assert this signal high for a minimum of one clock cycle, and de-assert the signal low prior to the end of the PR operation. The PR Controller IP core is ready to accept the next pr_start trigger event when the freeze signal is low. The PR Controller IP core ignores this signal during JTAG debug operations.</td>
</tr>
<tr>
<td>data[]</td>
<td>1, 8, 16, or 32</td>
<td>Input</td>
<td>Selectable input PR data bus width, either x1, x8, x16, or x32. Once a PR event triggers, the PR event is synchronous with the rising edge of the clk signal, whenever the data_valid signal is high, and the data_ready signal is high. The PR Controller IP core ignores this signal during JTAG debug operations.</td>
</tr>
<tr>
<td>data_valid</td>
<td>1</td>
<td>Input</td>
<td>A 0 to 1 transition on this port indicates the data[] port contains valid data. The PR Controller IP core ignores this signal during JTAG debug operations.</td>
</tr>
<tr>
<td>data_ready</td>
<td>1</td>
<td>Output</td>
<td>A 0 to 1 transition on this port indicates the PR Controller IP core is ready to read the valid data on the data[] port, whenever the data_valid signal asserts high. The data sender must stop sending valid data if this port is low. This signal deasserts low during JTAG debug operations.</td>
</tr>
<tr>
<td>status[2..0]</td>
<td>1</td>
<td>Output</td>
<td>A 3-bit output that indicates the status of PR events. When the IP detects an error (PR_ERROR, CRC_ERROR, or incompatible bitstream error), this signal latches high. This signal only resets at the beginning of the next PR event, when pr_start is high, and freeze is low. For example: 3'b000 – power-up or nreset asserts 3'b001 – PR_ERROR triggers 3'b010 – CRC_ERROR triggers</td>
</tr>
</tbody>
</table>
Table 27. **Avalon Memory-Mapped Slave Interface Ports**

These signals are available when **Enable Avalon memory-mapped slave interface** is **On**.

<table>
<thead>
<tr>
<th>Port Name</th>
<th>Width</th>
<th>Direction</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>avmm_slave_address</td>
<td>4</td>
<td>Input</td>
<td>Avalon memory-mapped address bus. The address bus is in the unit of Word addressing. The PR Controller IP core ignores this signal during JTAG debug operations.</td>
</tr>
<tr>
<td>avmm_slave_read</td>
<td>1</td>
<td>Input</td>
<td>Avalon memory-mapped read control. The PR Controller IP core ignores this signal during JTAG debug operations.</td>
</tr>
<tr>
<td>avmm_slave_readdata</td>
<td>32</td>
<td>Output</td>
<td>Avalon memory-mapped read data bus. The PR Controller IP core ignores this signal during JTAG debug operations.</td>
</tr>
<tr>
<td>avmm_slave_write</td>
<td>1</td>
<td>Input</td>
<td>Avalon memory-mapped write control. The PR Controller IP core ignores this signal during JTAG debug operations.</td>
</tr>
<tr>
<td>avmm_slave_writedata</td>
<td>32</td>
<td>Input</td>
<td>Avalon memory-mapped write data bus. The PR Controller IP core ignores this signal during JTAG debug operations.</td>
</tr>
<tr>
<td>avmm_slave_waitrequest</td>
<td>1</td>
<td>Output</td>
<td>Indicates that the IP is busy. Also indicates that the IP core is unable to respond to a read or write request. The IP core pulls this signal high during JTAG debug operations.</td>
</tr>
</tbody>
</table>

Table 28. **Interrupt Interface Ports**

These ports are available when **Enable interrupt interface** is **On**.

<table>
<thead>
<tr>
<th>Port Name</th>
<th>Width</th>
<th>Direction</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>irq</td>
<td>1</td>
<td>Output</td>
<td>The interrupt signal.</td>
</tr>
</tbody>
</table>

Table 29. **CRC BLOCK Interface**

These ports are available when **Use as Partial Reconfiguration Internal Host** is **Off**, or when you instantiate the **CRCBLOCK** manually for an internal host.

<table>
<thead>
<tr>
<th>Port Name</th>
<th>Width</th>
<th>Direction</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>crc_error_pin</td>
<td>1</td>
<td>Input</td>
<td>Available when you use the PR Controller IP core as an External Host. Connect this port to the dedicated CRC_ERROR pin of the FPGA undergoing partial reconfiguration.</td>
</tr>
</tbody>
</table>
Table 30. **PR Block Interface**

These options are available when **Use as Partial Reconfiguration Internal Host** is Off, or when you instantiate the **PRBLOCK** manually for an internal host.

<table>
<thead>
<tr>
<th>Port Name</th>
<th>Width</th>
<th>Direction</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>pr_ready_pin</td>
<td>1</td>
<td>Input</td>
<td>Connect this port to the dedicated <strong>PRREADY</strong> pin of the FPGA undergoing partial reconfiguration.</td>
</tr>
<tr>
<td>pr_error_pin</td>
<td>1</td>
<td>Input</td>
<td>Connect this port to the dedicated <strong>PRERROR</strong> pin of the FPGA undergoing partial reconfiguration.</td>
</tr>
<tr>
<td>pr_done_pin</td>
<td>1</td>
<td>Input</td>
<td>Connect this port to the dedicated <strong>PRDONE</strong> pin of the FPGA undergoing partial reconfiguration.</td>
</tr>
<tr>
<td>pr_request_pin</td>
<td>1</td>
<td>Output</td>
<td>Connect this port to the dedicated <strong>PRREQUEST</strong> pin of the FPGA undergoing partial reconfiguration.</td>
</tr>
<tr>
<td>pr_clk_pin</td>
<td>1</td>
<td>Output</td>
<td>Connect this port to the dedicated <strong>DCLK</strong> of the FPGA undergoing partial reconfiguration.</td>
</tr>
<tr>
<td>pr_data_pin[31..0]</td>
<td>16</td>
<td>Output</td>
<td>Connect this port to the dedicated <strong>DATA[31..0]</strong> pins of the FPGA undergoing partial reconfiguration.</td>
</tr>
</tbody>
</table>

### 2.3.6. Timing Specifications

The following timing diagram illustrates a successful PR operation with Partial Reconfiguration Controller Intel Arria 10/Cyclone 10 FPGA IP. The **status[2:0]** output signal indicates whether the operations passes or fails. The PR operation initiates upon assertion of the **pr_start** signal. Monitor the **status[]** signal to detect the end of the PR operation.

**Figure 54.** Partial Reconfiguration Controller Intel Arria 10/Cyclone 10 FPGA IP Timing Specifications

The following notes correspond to locations (1) through (7) in the timing diagram:

1. Assert **pr_start** signal high for a minimum of one clock cycle to initiate PR. Deassert **pr_start** before sending the last data.
2. **status[]** signal updates after **pr_start** is acknowledged. This signal changes during a PR operation if **CRC_ERROR**, **PR_ERROR**, or bitstream incompatibility error occurs.
3. **status[]** signal changes after a PR operation if **CRC_ERROR** asserts and no error occurs during the previous PR operation.
4. There is no requirement to assert the `data_valid` signal at the same time as the `pr_start` signal. Provide the `data[]`, and assert `data_valid`, when appropriate.

5. Either drive the `data_valid` signal low after sending the last data, or continue to assert `data_valid` high with dummy data on `data[]` until the IP reads the end of PR from `status[]`.

6. `data[]` transfers only when `data_valid` and `data_ready` assert on the same cycle. Do not drive new data on the data bus, when both `data_valid` and `data_ready` are not high.

7. The `data_ready` signal drives low after the PR IP Controller core receives the last data, or when the PR IP Controller cannot accept data.

### 2.3.7. PR Control Block and CRC Block Verilog HDL Manual Instantiation

The Partial Reconfiguration Controller Intel Arria 10/Cyclone 10 IP includes the PR control block. However, if you create your own custom logic to perform the function of the IP core, you can manually instantiate the control block to communicate with the FPGA system.

The following example instantiates a PR control block inside a top-level Intel Arria 10 PR project, `Chip_Top`, in Verilog HDL:

```verilog
Chip_Top:
module Chip_Top (
  //User I/O signals (excluding PR related signals)
  ...
  //PR interface and configuration signals declaration
  wire pr_request;
  wire pr_ready;
  wire pr_done;
  wire crc_error;
  wire dclk;
  wire [31:0] pr_data;

twentyfmp_rblock m_pr
  (.
   clk (dclk),
   .corectl (1'b1),
   .prrequest (pr_request),
   .data (pr_data),
   .error (pr_error),
   .ready (pr_ready),
   .done (pr_done)
  );

twentyfmp_crcblock m_crc
  (.
   clk (clk),
   .shiftdata (1'b1),
   .cerror (crc_error)
  );
endmodule
```

For more information about port connectivity for reading the Error Message Register (EMR), refer to the AN539: Test Methodology of Error Detection and Recovery using CRC.
2.3.8. PR Control Block and CRC Block VHDL Manual Instantiation

The following example shows manual instantiation of a PR control block inside your top-level Intel Arria 10 project, Chip_Top, in VHDL:

```vhdl
module Chip_Top is port (  
  --User I/O signals (excluding signals that relate to PR)  
  ..  
  ..  
  )  
-- Following shows the connectivity within the Chip_Top module  
Core_Top : Core_Top  
  port_map (  
    ..  
    ..  
  );  
m_pr : twentynm_prblock  
  port_map (  
    clk => dclk,  
    corectl =>'1', --1 - when using PR from inside  
    --0 - for PR from pins; You must also enable  
    -- the appropriate option in Quartus Prime settings  
    prrequest => pr_request,  
    data => pr_data,  
    error => pr_error,  
    ready => pr_ready,  
    done => pr_done  
  );  
m_crc : twentynm_crcblock  
  port_map (  
    shiftnld => '1', --If you want to read the EMR register when  
    clk => dummy_clk, --error occurs, refer to AN539 for the  
    --connectivity for this signal. If you only want  
    --to detect CRC errors, but plan to take no  
    --further action, you can tie the shiftnld  
    --signal to logical high.  
    crcerror => crc_error  
  );
```

**Note:** You are not required to connect a real clock source to `dummy_clk`, but you must connect `dummy_clk` to an I/O pin to avoid removal of this signal.

2.3.8.1. PR Control Block and CRC Block VHDL Component Declaration

The following example shows manual instantiation of the PR control block and the CRC block in your Intel Arria 10 PR design:

1. Use the code sample below, containing the component declaration in VHDL. This code performs the PR function from within the core (code block within `Core_Top`).

   ```vhdl
   module Chip_Top is port (  
      --User I/O signals (excluding signals that relate to PR)  
      ..  
      ..  
      )  
   -- Following shows the connectivity within the Chip_Top module  
   Core_Top : Core_Top  
   port_map (  
      ..  
   
   ```
m_pr : twentiesn_prblock
port map(
  clk => dclk,
  corectl => '1', --1 - when using PR from inside
  --0 - for PR from pins; You must also enable
  -- the appropriate option in Quartus Prime settings
  prrequest => pr_request,
  data => pr_data,
  error => pr_error,
  ready => pr_ready,
  done => pr_done
);

m_crc : twentiesn_crcblock
port map(
  shiftnld => '1', --If you want to read the EMR register when
  clk => dummy_clk, --error occurs, refer to AN539 for the
  --connectivity for this signal. If you only want
  --to detect CRC errors, but plan to take no
  --further action, you can tie the shiftnld
  --signal to logical high.
  crcerror => crc_error
);

Note: This VHDL example is adaptable for Verilog HDL instantiation.

2. Add additional ports to Core_Top to connect to both components.

3. Follow these rules when connecting the PR control block to the rest of your design:
   • Set the corectl signal to '1' (when using partial reconfiguration from core) or
     to '0' (when using partial reconfiguration from pins).
   • The corectl signal must match the Enable PR pins option setting in the
     Device and Pin Options dialog box (Assignments ➔ Device ➔ Device and
     Pin Options).
   • When performing partial reconfiguration from pins, the Fitter automatically
     assigns the PR unassigned pins. Assign all the dedicated PR pins using Pin
     Planner (Assignments ➔ Pin Planner) or Assignment Editor (Assignments
     ➔ Assignment Editor).
   • When performing partial reconfiguration from the core logic, connect the
     prblock signals to either core logic or I/O pins, excluding the dedicated
     programming pin, such as DCLK.

2.3.9. PR Control Block Signals

The following table lists the partial reconfiguration control block interface signals for
the Partial Reconfiguration Controller Intel Arria 10/Cyclone 10 FPGA IP:

Table 31. PR Control Block Interface Signals

<table>
<thead>
<tr>
<th>Signal</th>
<th>Width</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pr_data</td>
<td>[31:0]</td>
<td>Input</td>
<td>Carries the configuration bitstream.</td>
</tr>
<tr>
<td>pr_done</td>
<td>1</td>
<td>Output</td>
<td>Indicates that the PR process is complete.</td>
</tr>
</tbody>
</table>

continued...
<table>
<thead>
<tr>
<th>Signal</th>
<th>Width</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pr_ready</td>
<td>1</td>
<td>Output</td>
<td>Indicates that the control block is ready to accept PR data from the control logic.</td>
</tr>
<tr>
<td>pr_error</td>
<td>1</td>
<td>Output</td>
<td>Indicates a partial reconfiguration error.</td>
</tr>
<tr>
<td>pr_request</td>
<td>1</td>
<td>Input</td>
<td>Indicates that the PR process is ready to begin.</td>
</tr>
<tr>
<td>corectl</td>
<td>1</td>
<td>Input</td>
<td>Determines whether you are performing the partial reconfiguration internally, or through pins.</td>
</tr>
</tbody>
</table>

**Note:**
- You can specify a configuration width of 8, 16, or 32 bits, but the interface always uses 32 pins.
- All the inputs and outputs are asynchronous to the PR clock (clk), except data signal. data signal is synchronous to clk signal.
- PR clock must be free-running.
- data signal must be 0 while waiting for ready signal.

### 2.3.9.1. PR Control Block Signal Timing Diagrams

#### 2.3.9.1.1. Successful PR Session (Intel Arria 10 Example)

The following flow describes a successful Intel Arria 10 PR session:

1. Assert PR_REQUEST and wait for PR_READY; drive PR_DATA to 0.
2. The PR control block asserts PR_READY, asynchronous to clk.
3. Start sending Raw Binary File (.rbf) to the PR control block, with 1 valid word per clock cycle. On .rbf file transfer completion, drive PR_DATA to 0. The PR control block asynchronously asserts PR_DONE when the control block completes the reconfiguration operation. The PR control block deasserts PR_READY on configuration completion.
4. Deassert PR_REQUEST. The PR control block acknowledges the end of PR_REQUEST, and deasserts PR_DONE. The host can now initiate another PR session.
2.3.9.1.2. Unsuccessful PR Session with Configuration Frame Readback Error (Intel Arria 10 Example)

The following flow describes an Intel Arria 10 PR session with error in the EDCRC verification of a configuration frame readback:

1. The PR control block internally detects a CRC error.
2. The CRC control block then asserts CRC_ERROR.
3. The PR control block asserts the PR_ERROR.
4. The PR control block deasserts PR_READY, so that the host can withdraw the PR_REQUEST.
5. The PR control block deasserts CRC_ERROR and clears the internal CRC_ERROR signal to get ready for a new PR session. The host can now initiate another PR session.

Related Information
Raw Binary Programming File Byte Sequence Transmission Examples on page 29
2.3.9.1.3. Unsuccessful PR Session with PR_ERROR (Intel Arria 10 Example)

The following flow describes an Intel Arria 10 PR session with transmission error or configuration CRC error:

1. The PR control block asserts PR_ERROR.
2. The PR control block deasserts PR_READY, so that the host can withdraw PR_REQUEST.
3. The PR control block deasserts PR_ERROR to get ready for a new PR session. The host can now initiate another PR session.

Figure 57. Timing Diagram for Unsuccessful Intel Arria 10 PR Session with PR_ERROR

2.3.9.1.4. Late Withdrawal PR Session (Intel Arria 10 Example)

The following flow describes a late withdrawal Intel Arria 10 PR session:

1. The PR host can withdraw the request after the PR control block asserts PR_READY.
2. The PR control block deasserts PR_READY. The host can now initiate another PR session.

Figure 58. Timing Diagram for Late Withdrawal Intel Arria 10 PR Session

Note: The PR host can withdraw the request any time before the PR controller asserts PR_READY. Therefore, the PR host must not return until the PR control block asserts PR_READY. Provide at least 10 PR_CLK cycles after deassertion of PR_REQUEST, before requesting a new PR session.
2.3.10. Configuring an External Host for Intel Arria 10 or Intel Cyclone 10 GX Designs

When using external host configuration, the external host initiates partial reconfiguration, and monitors the PR status using the external PR dedicated pins during user mode. In this mode, the external host must respond appropriately to the handshake signals for successful partial reconfiguration. The external host writes the partial bitstream data from external memory into the Intel Arria 10 or Intel Cyclone 10 GX device. Co-ordinate system-level partial reconfiguration by ensuring that you prepare the correct PR region for partial reconfiguration. After reconfiguration, return the PR region into operating state.

To use an external host for your design:

1. Click Assignments ➤ Device ➤ Device & Pin Options.
2. Select the Enable PR Pins option in the Device & Pin Options dialog box. This option automatically creates the special partial reconfiguration pins, and defines the pins in the device pin-out. This option also automatically connects the pins to PR control block internal path.

   Note: If you do not select this option, you must use an internal or HPS host. You do not need to define pins in your design top-level entity.

3. Connect these top-level pins to the specific ports in the PR control block.

The following table lists the PR pins that automatically constrain when you turn on Enable PR Pins, and the specific PR control block port connection to the pin:

Table 32. Partial Reconfiguration Dedicated Pins

<table>
<thead>
<tr>
<th>Pin Name</th>
<th>Type</th>
<th>PR Control Block Port Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PR_REQUEST</td>
<td>Input</td>
<td>prrequest</td>
<td>Logic high on this pin indicates that the PR host is requesting partial reconfiguration.</td>
</tr>
<tr>
<td>PR_READY</td>
<td>Output</td>
<td>ready</td>
<td>Logic high on this pin indicates that the PR control block is ready to begin partial reconfiguration.</td>
</tr>
<tr>
<td>PR_DONE</td>
<td>Output</td>
<td>done</td>
<td>Logic high on this pin indicates that the partial reconfiguration is complete.</td>
</tr>
<tr>
<td>PR_ERROR</td>
<td>Output</td>
<td>error</td>
<td>Logic high on this pin indicates an error in the device during partial reconfiguration.</td>
</tr>
<tr>
<td>DATA[31:0]</td>
<td>Input</td>
<td>data</td>
<td>These pins provide connectivity for PR_DATA to transfer the PR bitstream to the PR controller.</td>
</tr>
<tr>
<td>DCLK</td>
<td>Input</td>
<td>clk</td>
<td>Receives synchronous PR_DATA.</td>
</tr>
</tbody>
</table>

Note:  
1. PR_DATA can be 8, 16, or 32-bits in width.  
2. Ensure that you connect the corectl port of the PR control block to 0.
Example 4. Verilog RTL for External Host PR

```verilog
module top(
    // PR control block signals
    input  logic        pr_clk,
    input  logic        pr_request,
    input  logic [31:0] pr_data,
    output logic        pr_error,
    output logic        pr_ready,
    output logic        pr_done,
    // User signals
    input  logic i1_main,
    input  logic i2_main,
    output logic o1
);

    // Instantiate the PR control block
    twentynm_prblock m_prblock
    (.
        clk(pr_clk),
        corectl(1'b0),
        prrequest(pr_request),
        data(pr_data),
        error(pr_error),
        ready(pr_ready),
        done(pr_done)
    );

    // PR Interface partition
    pr_v1 pr_inst(
        .i1(i1_main),
        .i2(i2_main),
        .o1(o1)
    );

endmodule
```

Example 5. VHDL RTL for External Host PR

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity top is
    port(
        -- PR control block signals
        pr_clk: in std_logic;
        pr_request: in std_logic;
        pr_data: in std_logic_vector(31 downto 0);
        pr_error: out std_logic;
        pr_ready: out std_logic;
        pr_done: out std_logic;
        -- User signals
        i1_main: in std_logic;
        i2_main: in std_logic;
        o1: out std_logic
    );
end top;

architecture behav of top is
    component twentynm_prblock is
        port(
            clk: in std_logic;
            corectl: in std_logic;
            prrequest: in std_logic;
            data: in std_logic_vector(31 downto 0);
```
error: out std_logic;
ready: out std_logic;
done: out std_logic
);
end component;

component pr_v1 is
port(
i1: in std_logic;
i2: in std_logic;
o1: out std_logic
);
end component;
signal pr_gnd : std_logic;
begins
pr_gnd <= '0';

-- Instantiate the PR control block
m_prblock: twentynm_prblock port map
(
    pr_clk,
    pr_gnd,
    pr_request,
    pr_data,
    pr_error,
    pr_ready,
    pr_done
);

-- PR Interface partition
pr_inst : pr_v1 port map
(
    i1_main,
    i2_main,
    o1
);
end behav;

2.4. Partial Reconfiguration External Configuration Controller Intel FPGA IP

The Partial Reconfiguration External Configuration Controller Intel FPGA IP supports partial reconfiguration via an external source.

When using external configuration, you must connect all the top-level ports of the Partial Reconfiguration External Configuration Controller Intel FPGA IP to the `pr_request` and `status` pins, to allow the handshaking of the host with SDM from the Intel Stratix 10 or Intel Agilex core. The SDM determines which types of configuration pins to use, according your `MSEL` setting.
2.4.1. Parameters

The Partial Reconfiguration External Configuration Controller Intel FPGA IP supports customization of the following parameters.

Table 33. Partial Reconfiguration External Configuration Controller Parameter Settings

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable Busy Interface</td>
<td>On/Off</td>
<td>Allows you to Enable or Disable the Busy interface, which asserts a signal to indicate that PR processing is in progress during external configuration.</td>
</tr>
</tbody>
</table>

2.4.2. Ports

The Partial Reconfiguration External Configuration Controller Intel FPGA IP includes the following interface ports.
### Table 34. Ports

<table>
<thead>
<tr>
<th>Port Name</th>
<th>Width</th>
<th>Direction</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>pr_request</td>
<td>1</td>
<td>Input</td>
<td>Indicates that the PR process is ready to begin. The signal is a conduit not synchronous to any clock signal.</td>
</tr>
<tr>
<td>pr_error</td>
<td>1</td>
<td>Output</td>
<td>Indicates a partial reconfiguration error. The signal is a conduit not synchronous to any clock signal.</td>
</tr>
<tr>
<td>pr_done</td>
<td>1</td>
<td>Output</td>
<td>Indicates that the PR process is complete. The signal is a conduit not synchronous to any clock signal.</td>
</tr>
<tr>
<td>start_addr</td>
<td>1</td>
<td>Input</td>
<td>Specifies the start address of PR data in Active Serial Flash. You enable this signal by selecting either Avalon-ST or Active Serial for the Enable Avalon-ST Pins or Active Serial Pins parameter. The signal is a conduit not synchronous to any clock signal.</td>
</tr>
<tr>
<td>reset</td>
<td>1</td>
<td>Input</td>
<td>Active high, synchronous reset signal.</td>
</tr>
<tr>
<td>out_clock</td>
<td>1</td>
<td>Output</td>
<td>Clock source that generates from an internal oscillator.</td>
</tr>
<tr>
<td>busy</td>
<td>1</td>
<td>Output</td>
<td>The IP asserts this signal to indicate PR data transfer in progress. You enable this signal by selecting Enable for the Enable busy interface parameter.</td>
</tr>
</tbody>
</table>

#### 2.4.3. Configuring an External Host for Intel Stratix 10 or Intel Agilex Designs

You can optionally use an external host to write the partial bitstream data from external memory into the Intel Stratix 10 or Intel Agilex device. When using external host configuration, the external host initiates partial reconfiguration by asserting the pr_request signal. The external host monitors the PR status through the pr_done and pr_error signals.

The external host must respond appropriately to the handshake signals for successful partial reconfiguration. Co-ordinate system-level partial reconfiguration by ensuring that you prepare the correct PR region for partial reconfiguration. After reconfiguration, return the PR region into operating state.

To configure an external host for Intel Stratix 10 or Intel Agilex designs, follow these steps:

1. Parameterize and generate the Partial Reconfiguration External Configuration Controller Intel FPGA IP, as Generating IP Cores (Intel Quartus Prime Pro Edition) on page 122 describes.
2. Connect the Partial Reconfiguration External Configuration Controller pr_request, pr_done, and pr_error signals to top-level pins for control and monitor by the external host. You can assign the pin location by clicking Assignments ➤ Pin Planner.
3. Click Assignments ➤ Device, and then click the Device & Pin Options button.
4. In the Category list, click Configuration.
5. For the Configuration scheme, select the scheme that matches with your full device configuration. For example, if your full device configuration uses the AVSTx32 scheme, the PR configuration must use AVSTx32. This option automatically reserves dedicated Avalon streaming configuration pins for partial reconfiguration during user mode. The pins are exactly same as the Avalon streaming pins that you use for full device configuration.
The following table describes the PR pins that the external host uses. The PR streaming to Avalon streaming pins must conform to the Avalon streaming specification for data transfer with backpressure.

<table>
<thead>
<tr>
<th>Pin Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pr_request</td>
<td>Input</td>
<td>User-assigned port connected to Partial Reconfiguration External Configuration Controller IP. Logic high on this pin indicates that the PR host is requesting partial reconfiguration.</td>
</tr>
<tr>
<td>pr_done</td>
<td>Output</td>
<td>User-assigned port connected to Partial Reconfiguration External Configuration Controller IP. Logic high on this pin indicates that the partial reconfiguration is complete.</td>
</tr>
<tr>
<td>pr_error</td>
<td>Output</td>
<td>User-assigned port connected to Partial Reconfiguration External Configuration Controller IP. Logic high on this pin indicates an error in the device during partial reconfiguration.</td>
</tr>
<tr>
<td>avst_data:</td>
<td>Input</td>
<td>These pins provide connectivity for the external host to transfer the PR bitstream to the SDM. The <code>avstx8</code> data pins are part of the SDM I/O. <code>avstx16</code> and <code>avstx32</code> data pins are from I/O 48 bank 3A.</td>
</tr>
<tr>
<td></td>
<td>avstx8 - [7:0]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>avstx16 - [15:0]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>avstx32 - [31:0]</td>
<td></td>
</tr>
<tr>
<td>avst_clk</td>
<td>Input</td>
<td>Clocks the Avalon streaming interfaces. <code>avst_data</code> and <code>avst_valid</code> are synchronous with <code>avst_clk</code>. The <code>avstx8</code> clk pin is part of the SDM I/O. <code>avstx16</code> and <code>avstx32</code> are from I/O 48 bank 3A.</td>
</tr>
<tr>
<td>avst_valid</td>
<td>Input</td>
<td>Logic high on this pin indicates the data in <code>avst_data</code> is valid data. The <code>avstx8</code> data pins are part of the SDM I/O. <code>avstx16</code> and <code>avstx32</code> data pins are from I/O 48 bank 3A.</td>
</tr>
<tr>
<td>avst_ready</td>
<td>Output</td>
<td>Logic high on this pin indicates the SDM is ready to accept data from an external host. This output is part of the SDM I/O.</td>
</tr>
</tbody>
</table>

2.5. Partial Reconfiguration Region Controller Intel FPGA IP

The Partial Reconfiguration Region Controller Intel FPGA IP provides a standard interface through the Freeze Control block that controls handshaking with the PR region. The PR handshake ensures that PR region transactions complete before freeze of the interface.
2.5.1. Registers

The Partial Reconfiguration Region Controller IP core performs the following operations in a partial reconfiguration:

<table>
<thead>
<tr>
<th>IP Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Freeze Control and Status Register</td>
<td>Freeze status register that generates the freeze output signal.</td>
</tr>
<tr>
<td>Freeze Control Block</td>
<td>Performs PR handshaking and resets the PR region.</td>
</tr>
<tr>
<td>Conduit Splitter</td>
<td>Connects the controller’s freeze signal to one or more Freeze Bridge components. Receives the freeze signal from the Freeze Control Block, and assigns the freeze input signal to one or more freeze output signals.</td>
</tr>
<tr>
<td>Conduit Merger</td>
<td>Connects the illegal_request signal from one or more Freeze Bridge components to the PR Region Controller. The illegal_request is a single-bit output signal from the Freeze Bridge. Conduit Merger concatenates the single-bit signal from multiple Freeze Bridges into a multi-bit bus. The Conduit Merger then connects the bus to the Freeze Control Block.</td>
</tr>
</tbody>
</table>
Figure 62. Freeze Control Block PR Handshake Timing

- **Unbounded time between stop_req and stop_ack**
- **stop_req** is deasserted when stop_ack is asserted
- **start_req is asserted** when freeze is deasserted

- **region_reset**
- **start_req** is deasserted when start_ack is asserted
- **start_ack**
- **region_reset** is fully user controlled
- **Unbounded time between start_req and start_ack** when start_ack is asserted
Figure 63. Register States and Programming Model

Freeze Controller States

- Controller detects "1" on freeze_csr_ctrl register freeze_req
  - Controller asserts stop_req to stop executing current PR region content
  - PR region asserts stop_ack?
    - No
    - Yes
      - Controller deasserts stop_req and asserts freeze to freeze the bridges and other region outputs
  - Controller asserts freeze_csr_status register freeze_status bit
  - Write '1' to reset_req bit of freeze_csr_ctrl to assert PR reset
  - Wait X cycles before writing '0' to freeze_csr_ctrl reset_req bit to deassert PR region reset
  - Freeze_csr_ctrl register freeze_bit deasserted?
    - No
    - Yes
      - Controller asserts start_req to start new persona
      - Write '1' to freeze_csr_ctrl freeze_req bit
      - freeze_csr_status freeze_status bit '1'? No
      - Yes
        - Do Partial Reconfiguration thru HPS FPGA manager or PR Controller IP
        - Wait X cycles before writing '0' to freeze_csr_ctrl reset_req bit to deassert PR region reset
    - freeze_csr_status unfreeze_status bit '1'?
      - No
      - Yes
        - Partial Reconfiguration operation complete

Software Programming Flow

- Confirm all freeze_csr_status register bits read '0'
  - Write '1' to freeze_csr_ctrl freeze_req bit
  - freeze_csr_status freeze_status bit '1'?
    - No
    - Yes
      - Write '1' to reset_req bit of freeze_csr_ctrl to assert PR reset
      - Do Partial Reconfiguration thru HPS FPGA manager or PR Controller IP
      - Wait X cycles before writing '0' to freeze_csr_ctrl reset_req bit to deassert PR region reset
      - freeze_csr_status unfreeze_status bit '1'?
        - No
        - Yes
          - Partial Reconfiguration operation complete
### Table 37. Register Map

<table>
<thead>
<tr>
<th>Name</th>
<th>Address Offset</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>freeze_csr_status</td>
<td>0x00</td>
<td>Read-Only</td>
<td>Freeze status register.</td>
</tr>
<tr>
<td>csr_ctrl</td>
<td>0x01</td>
<td>Read or Write</td>
<td>Control register to enable and disable freeze.</td>
</tr>
<tr>
<td>freeze_illegal_req</td>
<td>0x02</td>
<td>Read or Write</td>
<td>High on any bit indicates an illegal request during the freeze state.</td>
</tr>
<tr>
<td>freeze_reg_version</td>
<td>0x03</td>
<td>Read-Only</td>
<td>Read-only version register. This register is currently 0xAD000003.</td>
</tr>
</tbody>
</table>

### Table 38. freeze_csr_status

<table>
<thead>
<tr>
<th>Bit</th>
<th>Fields</th>
<th>Access</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31:2</td>
<td>Reserved</td>
<td>N/A</td>
<td>0x0</td>
<td>Reserved bits. Reading these bits always returns zeros.</td>
</tr>
<tr>
<td>1</td>
<td>unfreeze_status</td>
<td>R</td>
<td>0</td>
<td>Hardware sets this bit to 1 after the PR region returns start_ack to indicate successful start of the persona. Hardware clears this bit to 0 when the unfreeze_req bit is low. This bit is 1 when bridges and other PR region outputs release from reset.</td>
</tr>
<tr>
<td>0</td>
<td>freeze_status</td>
<td>R</td>
<td>0</td>
<td>Hardware sets this bit to 1 after the PR region returns the stop_ack signal to indicate that the PR region is ready to enter the frozen state. Hardware clears this bit to 0 when the freeze_req bit is low. This bit is 0 when bridges and other PR region outputs release from reset.</td>
</tr>
</tbody>
</table>

### Table 39. freeze_csr_ctrl

<table>
<thead>
<tr>
<th>Bit</th>
<th>Fields</th>
<th>Access</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31:3</td>
<td>Reserved</td>
<td>N/A</td>
<td>0x0</td>
<td>Reserved bits. Reading these bits always returns zeros.</td>
</tr>
<tr>
<td>2</td>
<td>unfreeze_req</td>
<td>R/W</td>
<td>0</td>
<td>Write 1 to this bit to request unfreezing the PR region interfaces. Hardware clears this bit after unfreeze_status is high. Write 0 to this bit to terminate the unfreeze request. Do not assert this bit and the freeze_req bit at the same time. If both freeze_req and unfreeze_req assert at the same time, it is an invalid operation.</td>
</tr>
<tr>
<td>1</td>
<td>reset_req</td>
<td>R/W</td>
<td>0</td>
<td>Write 1 to start resetting the PR persona. Write 0 to stop resetting the PR persona.</td>
</tr>
<tr>
<td>0</td>
<td>freeze_req</td>
<td>R/W</td>
<td>0</td>
<td>Write 1 to this bit to start freezing the PR region interfaces. Hardware clears this bit after freeze_status is high.</td>
</tr>
<tr>
<td>Bit</td>
<td>Fields</td>
<td>Access</td>
<td>Default Value</td>
<td>Description</td>
</tr>
<tr>
<td>-----</td>
<td>------------</td>
<td>--------</td>
<td>---------------</td>
<td>---------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Write 0 to this bit to terminate the freeze request if the PR region never returns stop_ack after this bit asserts. Do not assert this bit and the unfreeze_req bit at the same time. Asserting freeze_req and unfreeze_req simultaneously is an invalid operation.</td>
</tr>
</tbody>
</table>

**Table 40. freezeIllegal_request**

<table>
<thead>
<tr>
<th>Bit</th>
<th>Fields</th>
<th>Access</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31:n</td>
<td>Reserved</td>
<td>N/A</td>
<td>0x0</td>
<td>Reserved bits. Reading these bits always returns zeros.</td>
</tr>
<tr>
<td>n-1:0</td>
<td>illegal_request</td>
<td>R/W</td>
<td>0</td>
<td>High on any bit of this bus indicates a read or write issue by a static region master when an Avalon memory-mapped slave freeze bridge is in the freeze state. Identify which freeze bridge has an illegal request by checking each bit on the bus. For example, when illegal_request bit 2 is high, an illegal request occurred in the freeze bridge that connects to interface freeze_conduit_in2. This bus triggers the interrupt signal. Write 1 to clear this bit. n is the number of bridges.</td>
</tr>
</tbody>
</table>

**Table 41. freeze_reg_version**

<table>
<thead>
<tr>
<th>Bit</th>
<th>Fields</th>
<th>Access</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31:0</td>
<td>Version Register</td>
<td>Read-Only</td>
<td>AD000003</td>
<td>This register bit indicates the CSR register version number. Currently the CSR register is version 0xAD000003.</td>
</tr>
</tbody>
</table>

### 2.5.2. Parameters

The Partial Reconfiguration Region Controller IP core supports customization of the following parameters.

**Table 42. Partial Reconfiguration Region Controller Parameter Settings**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable Avalon-MM CSR register</td>
<td>On/Off</td>
<td>On</td>
<td>Enables Avalon memory-mapped CSR registers in the PR region controller. Disable this option to expose a conduit interface and not instantiate the CSR block.</td>
</tr>
<tr>
<td>Enable interrupt port for illegal request</td>
<td>On/Off</td>
<td>On</td>
<td>Enables the interrupt port for illegal operations in the PR region controller.</td>
</tr>
</tbody>
</table>

continued...
### 2.5.3. Ports

The Partial Reconfiguration Region Controller IP has the following ports.

#### Table 43. Freeze CSR Block Ports

These ports are available when **Enable Avalon Memory-Mapped CSR Register** is **On**.

<table>
<thead>
<tr>
<th>Port</th>
<th>Width</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clock_clk</td>
<td>1</td>
<td>Input</td>
<td>IP core input clock.</td>
</tr>
</tbody>
</table>

**Reset**

<table>
<thead>
<tr>
<th>Port</th>
<th>Width</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reset_reset</td>
<td>1</td>
<td>Input</td>
<td>Synchronous reset.</td>
</tr>
<tr>
<td>avl csr_addr</td>
<td>2</td>
<td>Input</td>
<td>Avalon memory-mapped address bus. The address bus is in word addressing.</td>
</tr>
<tr>
<td>avl csr_read</td>
<td>1</td>
<td>Input</td>
<td>Avalon memory-mapped read control to CSR.</td>
</tr>
<tr>
<td>avl csr_write</td>
<td>1</td>
<td>Input</td>
<td>Avalon memory-mapped write control to CSR.</td>
</tr>
</tbody>
</table>
### Table 44. Freeze Control Block Ports

<table>
<thead>
<tr>
<th>Port</th>
<th>Width</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>avl_csr_writedata</td>
<td>32</td>
<td>Input</td>
<td>Avalon memory-mapped write data bus to CSR.</td>
</tr>
<tr>
<td>avl_csr_readdata</td>
<td>32</td>
<td>Output</td>
<td>Avalon memory-mapped read data bus from CSR.</td>
</tr>
<tr>
<td>interrupt_sender_irq</td>
<td>1</td>
<td>output</td>
<td>Trigger by illegal read or illegal write.</td>
</tr>
</tbody>
</table>

### Table 45. Conduit Splitter and Merger Interface Ports

<table>
<thead>
<tr>
<th>Signal</th>
<th>Width</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bridge_freeze0_freeze</td>
<td>1</td>
<td>Output</td>
<td>This output connects to the freeze input signal of a freeze bridge IP or to control other freeze logic. (Multiple interfaces generate according to the number of freeze interfaces)</td>
</tr>
<tr>
<td>bridge_freeze0_illegal_request</td>
<td>1</td>
<td>Input</td>
<td>This input connects to the illegal_request output signal from an instance of the Freeze Bridge IP.</td>
</tr>
</tbody>
</table>
Figure 65. Partial Reconfiguration Region Controller Interface Ports (Control and Status Register Block Enabled)

```
pr_region_controller_0
```

Figure 66. Partial Reconfiguration Region Controller Interface Ports (Control and Status Register Block Disabled)

```
pr_region_controller_0
```

2.6. Avalon Memory-Mapped Partial Reconfiguration Freeze Bridge IP

The Avalon Memory-Mapped Partial Reconfiguration Freeze Bridge Intel FPGA IP freezes a PR region Avalon memory-mapped interface when the freeze input signal is high. It is recommended that each Avalon memory-mapped interface to a PR region use an instance of the Freeze Bridge IP.

Figure 67. Avalon Memory-Mapped Partial Reconfiguration Freeze Bridge IP

<table>
<thead>
<tr>
<th>Table 46. Read and Write Request to PR Region Avalon Memory-Mapped Slave Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Interface Connection</strong></td>
</tr>
</tbody>
</table>
| Read request to Avalon memory-mapped slave interface in PR region | 1. During the freeze state, any read transaction responds with bogus data <h'DEADBEEF'>. The corresponding freeze_illegal_request register bit sets.  
2. During the freeze state, readrequest, writerequest, waitrequest, beginbursttransfer, lock, and debugaccess signals in the PR region interface tie low.  
3. The Avalon memory-mapped slave response signal constantly returns 2'b10, to indicate an unsuccessful transaction from an endpoint slave.  
4. If you disable Enable Freeze port from PR region, the IP generates no responses. |
| Write request to slave interface in PR region | 1. The Freeze Bridge ignores any write transactions during the freeze state. The Freeze Bridge pulls the waitrequest, beginbursttransfer, lock and debugaccess signals low. The IP sets the corresponding freeze_illegal_request register bit.  
2. The Avalon memory-mapped slave response signal updates with 2'b10 to indicate an unsuccessful transaction from an endpoint slave.  
3. If you disable Enable Freeze port from PR region, the IP generates no responses. |

<table>
<thead>
<tr>
<th>Table 47. Read and Write Request from PR Region Avalon-MM Master Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Interface Connection</strong></td>
</tr>
</tbody>
</table>
| Read/Write request from Avalon-MM master interface in PR region (old or new persona) | 1. During the freeze state the IP ignores the read and write signals from the PR region.  
2. The read and write signals to the static region deassert. |
Table 48.  **Avalon-MM Partial Reconfiguration Freeze Bridge Signal Behavior**

The table below summarizes the Avalon interface output signal behavior when the Freeze Bridge is in a frozen state. When not frozen, all signals are just pass-through.

<table>
<thead>
<tr>
<th>Signal</th>
<th>Avalon-MM Slave Bridge</th>
<th>Avalon-MM Master Bridge</th>
</tr>
</thead>
<tbody>
<tr>
<td>write</td>
<td>'b0 (tie low)</td>
<td>'b0 (tie low)</td>
</tr>
<tr>
<td>read</td>
<td>'b0 (tie low)</td>
<td>'b0 (tie low)</td>
</tr>
<tr>
<td>address</td>
<td>Pass through</td>
<td>Pass through</td>
</tr>
<tr>
<td>writedata</td>
<td>Pass through</td>
<td>Pass through</td>
</tr>
<tr>
<td>readdata</td>
<td>Return &lt;h'DEADBEEF&gt; always</td>
<td>Pass through</td>
</tr>
<tr>
<td>byteenable</td>
<td>Pass through</td>
<td>Pass through</td>
</tr>
<tr>
<td>burstcount</td>
<td>Pass through</td>
<td>Pass through</td>
</tr>
<tr>
<td>beginbursttransfer</td>
<td>'b0 (tie low)</td>
<td>'b0 (tie low)</td>
</tr>
<tr>
<td>debugaccess</td>
<td>'b0 (tie low)</td>
<td>'b0 (tie low)</td>
</tr>
<tr>
<td>readdatavalid</td>
<td>Return 'b1 when there is a request, else 'b0</td>
<td>Pass through</td>
</tr>
<tr>
<td>waitrequest</td>
<td>Return 'b1 when there is a request, else 'b0</td>
<td>'b0 (tie low)</td>
</tr>
<tr>
<td>response</td>
<td>Return 'b10 always</td>
<td>Pass through</td>
</tr>
<tr>
<td>lock</td>
<td>'b0 (tie low)</td>
<td>'b0 (tie low)</td>
</tr>
<tr>
<td>writeresponsevalid</td>
<td>Return 'b1 when there is a request, else 'b0</td>
<td>Pass through</td>
</tr>
</tbody>
</table>

2.6.1. **Parameters**

The Avalon Memory-Mapped Partial Reconfiguration Freeze Bridge IP core supports customization of the following parameters.

Table 49.  **Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PR region interface Type</td>
<td>Avalon-MM Slave/Avalon-MM Master</td>
<td>Specifies the interface type for interfacing the PR region with the Freeze Bridge.</td>
</tr>
<tr>
<td>Enable Freeze port from PR region</td>
<td>On/Off</td>
<td>Enables the freeze port that freezes all the outputs of each PR region to a known constant value. Freezing prevents the signal receivers in the static region from receiving undefined signals during the partial reconfiguration process. The freeze of a bridge is the logical OR of this signal from the PR region, and the freeze from the PR region controller.</td>
</tr>
<tr>
<td>Enable the bridge to track unfinished transaction</td>
<td>On/Off</td>
<td>Enables the bridge to track unfinished transactions before freezing the Avalon interface. Turn on this option when there is no custom logic to stop the Avalon transaction between the PR region and the static region. If you do not need this feature, disable this option to reduce the size of the IP.</td>
</tr>
<tr>
<td>Enabled Avalon Interface Signal</td>
<td>Yes/No</td>
<td>Enable (Yes) or disable (No) specific optional Freeze Bridge interface ports.</td>
</tr>
<tr>
<td>Address width</td>
<td>&lt;1-64&gt;</td>
<td>Address width in bits.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Values</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Symbol width</td>
<td>&lt;number&gt;</td>
<td>Data symbol width in bits. The symbol width should be 8 for byte-oriented interfaces.</td>
</tr>
<tr>
<td>Number of symbols</td>
<td>&lt;number&gt;</td>
<td>Number of symbols per word.</td>
</tr>
<tr>
<td>Burstcount width</td>
<td>&lt;number&gt;</td>
<td>The width of the burst count in bits.</td>
</tr>
<tr>
<td>Linewrap burst</td>
<td>On/Off</td>
<td>When On, the address for bursts wraps instead of incrementing. With a wrapping burst, when the address reaches a burst boundary, the address wraps back to the previous burst boundary. Consequently, the IP uses only the low order bits for addressing.</td>
</tr>
<tr>
<td>Constant burst behavior</td>
<td>On/Off</td>
<td>When On, memory bursts are constant.</td>
</tr>
<tr>
<td>Burst on burst boundaries only</td>
<td>On/Off</td>
<td>When On, memory bursts are aligned to the address size.</td>
</tr>
<tr>
<td>Maximum pending reads</td>
<td>&lt;number&gt;</td>
<td>The maximum number of pending reads that the slave can queue.</td>
</tr>
<tr>
<td>Maximum pending writes</td>
<td>&lt;number&gt;</td>
<td>The maximum number of pending writes that the slave can queue.</td>
</tr>
<tr>
<td>Fixed read latency (cycles)</td>
<td>&lt;number&gt;</td>
<td>Sets the read latency for fixed-latency slaves. Not useful on interfaces that include the readdatavalid signal.</td>
</tr>
<tr>
<td>Fixed read wait time (cycles)</td>
<td>&lt;number&gt;</td>
<td>For master interfaces that do not use the waitrequest signal. The read wait time indicates the number of cycles before the master responds to a read. The timing is as if the master asserted waitrequest for this number of cycles.</td>
</tr>
<tr>
<td>Fixed write wait time (cycles)</td>
<td>&lt;number&gt;</td>
<td>For master interfaces that do not use the waitrequest signal. The write wait time indicates the number of cycles before the master accepts a write.</td>
</tr>
<tr>
<td>Address type</td>
<td>WORDS/SYMBOLES</td>
<td>Sets slave interface address type to symbols or words.</td>
</tr>
</tbody>
</table>
### 2.6.2. Interface Ports

The Avalon Memory-Mapped Partial Reconfiguration Freeze Bridge IP core has the following interface ports.

<table>
<thead>
<tr>
<th>Port</th>
<th>Width</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clock</td>
<td>1</td>
<td>Input</td>
<td>Input clock for the IP.</td>
</tr>
<tr>
<td>reset_n</td>
<td>1</td>
<td>Input</td>
<td>Synchronous reset for the IP.</td>
</tr>
<tr>
<td>Port</td>
<td>Width</td>
<td>Direction</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------</td>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>freeze_conduit_freeze</td>
<td>1</td>
<td>Input</td>
<td>When this signal is high, the bridge handles any current transaction properly then freezes the Avalon memory-mapped PR interfaces.</td>
</tr>
<tr>
<td>freeze_conduit_illegal_request</td>
<td>1</td>
<td>Output</td>
<td>High on this bus indicates that an illegal request was issued to the bridge during the freeze state.</td>
</tr>
<tr>
<td>pr_freeze_pr_freeze</td>
<td>1</td>
<td>Input</td>
<td>Enabled freeze port coming from the PR region.</td>
</tr>
</tbody>
</table>

Table 51. Avalon Memory-Mapped Slave to PR Region Master Interface Ports

<table>
<thead>
<tr>
<th>Port</th>
<th>Width</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>slv_bridge_to_pr_read</td>
<td>1</td>
<td>Output</td>
<td>Optional Avalon memory-mapped slave bridge to PR region read port.</td>
</tr>
<tr>
<td>slv_bridge_to_pr_waitrequest</td>
<td>1</td>
<td>Input</td>
<td>Optional Avalon memory-mapped slave bridge to PR region waitrequest port.</td>
</tr>
<tr>
<td>slv_bridge_to_pr_write</td>
<td>1</td>
<td>Output</td>
<td>Optional Avalon memory-mapped slave bridge to PR region write port.</td>
</tr>
<tr>
<td>slv_bridge_to_pr_address</td>
<td>32</td>
<td>Output</td>
<td>Optional Avalon memory-mapped slave bridge to PR region address port.</td>
</tr>
<tr>
<td>slv_bridge_to_pr_byteenable</td>
<td>4</td>
<td>Output</td>
<td>Optional Avalon memory-mapped slave bridge to PR region byteenable port.</td>
</tr>
<tr>
<td>slv_bridge_to_pr_writedata</td>
<td>32</td>
<td>Output</td>
<td>Optional Avalon memory-mapped slave bridge to PR region writedata port.</td>
</tr>
<tr>
<td>slv_bridge_to_pr_readdata</td>
<td>32</td>
<td>Input</td>
<td>Optional Avalon memory-mapped slave bridge to PR region readdata port.</td>
</tr>
<tr>
<td>slv_bridge_to_pr_burstcount</td>
<td>3</td>
<td>Output</td>
<td>Optional Avalon memory-mapped slave bridge to PR region burstcount port.</td>
</tr>
<tr>
<td>slv_bridge_to_pr_readdatavalid</td>
<td>1</td>
<td>Input</td>
<td>Optional Avalon memory-mapped slave bridge to PR region readdatavalid port.</td>
</tr>
<tr>
<td>slv_bridge_to_pr_beginbursttransfer</td>
<td>1</td>
<td>Output</td>
<td>Optional Avalon-MM slave bridge to PR region beginbursttransfer port.</td>
</tr>
<tr>
<td>slv_bridge_to_pr_debugaccess</td>
<td>1</td>
<td>Output</td>
<td>Optional Avalon memory-mapped slave bridge to PR region debugaccess port.</td>
</tr>
<tr>
<td>slv_bridge_to_pr_response</td>
<td>2</td>
<td>Input</td>
<td>Optional Avalon memory-mapped slave bridge to PR region response port.</td>
</tr>
<tr>
<td>slv_bridge_to_pr_lock</td>
<td>1</td>
<td>Output</td>
<td>Optional Avalon-MM slave bridge to PR region lock port.</td>
</tr>
<tr>
<td>slv_bridge_to_pr_writeresponsevalid</td>
<td>1</td>
<td>Input</td>
<td>Optional Avalon memory-mapped slave bridge to PR region writeresponsevalid port.</td>
</tr>
</tbody>
</table>
Table 52. Avalon Memory-Mapped Slave to Static Region Master Interface Ports

<table>
<thead>
<tr>
<th>Port</th>
<th>Width</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>slv_bridge_to_sr_read</td>
<td>1</td>
<td>Input</td>
<td>Avalon memory-mapped slave bridge to static region read port.</td>
</tr>
<tr>
<td>slv_bridge_to_sr_writedata</td>
<td>32</td>
<td>Input</td>
<td>Avalon memory-mapped slave bridge to static region writedata port.</td>
</tr>
<tr>
<td>slv_bridge_to_sr_waitrequest</td>
<td>1</td>
<td>Output</td>
<td>Avalon memory-mapped slave bridge to static region waitrequest port.</td>
</tr>
<tr>
<td>slv_bridge_to_sr_write</td>
<td>1</td>
<td>Input</td>
<td>Avalon memory-mapped slave bridge to static region write port.</td>
</tr>
<tr>
<td>slv_bridge_to_sr_address</td>
<td>32</td>
<td>Input</td>
<td>Avalon memory-mapped slave bridge to static region address port.</td>
</tr>
<tr>
<td>slv_bridge_to_sr_byteenable</td>
<td>4</td>
<td>Input</td>
<td>Avalon memory-mapped slave bridge to static region byteenable port.</td>
</tr>
<tr>
<td>slv_bridge_to_sr_burstcount</td>
<td>3</td>
<td>Input</td>
<td>Avalon memory-mapped slave bridge to static region burstcount port.</td>
</tr>
<tr>
<td>slv_bridge_to_sr_beginbursttransfer</td>
<td>1</td>
<td>Input</td>
<td>Avalon memory-mapped slave bridge to static region beginbursttransfer port.</td>
</tr>
<tr>
<td>slv_bridge_to_sr_debugaccess</td>
<td>1</td>
<td>Input</td>
<td>Avalon-MM slave bridge to static region debugaccess port.</td>
</tr>
<tr>
<td>slv_bridge_to_sr_response</td>
<td>2</td>
<td>Output</td>
<td>Avalon memory-mapped slave bridge to static region response port.</td>
</tr>
<tr>
<td>slv_bridge_to_sr_lock</td>
<td>1</td>
<td>Input</td>
<td>Avalon memory-mapped slave bridge to static region lock port.</td>
</tr>
<tr>
<td>slv_bridge_to_sr_writereponsevalid</td>
<td>1</td>
<td>Output</td>
<td>Avalon memory-mapped slave bridge to static region writereponsevalid port.</td>
</tr>
</tbody>
</table>

Note: Same setting as Avalon memory-mapped master to PR region slave interface.

Table 53. Avalon Memory-Mapped Master to PR Region Slave Interface Ports

<table>
<thead>
<tr>
<th>Port</th>
<th>Width</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mst_bridge_to_pr_read</td>
<td>1</td>
<td>Input</td>
<td>Optional Avalon memory-mapped master bridge to PR region read port.</td>
</tr>
<tr>
<td>mst_bridge_to_pr_writedata</td>
<td>32</td>
<td>Input</td>
<td>Optional Avalon memory-mapped master bridge to PR region writedata port.</td>
</tr>
<tr>
<td>mst_bridge_to_pr_waitrequest</td>
<td>1</td>
<td>Output</td>
<td>Optional Avalon memory-mapped master bridge to PR region waitrequest port.</td>
</tr>
<tr>
<td>mst_bridge_to_pr_write</td>
<td>1</td>
<td>Input</td>
<td>Optional Avalon memory-mapped master bridge to PR region write port.</td>
</tr>
<tr>
<td>mst_bridge_to_pr_address</td>
<td>32</td>
<td>Input</td>
<td>Optional Avalon memory-mapped master bridge to PR region address port.</td>
</tr>
<tr>
<td>mst_bridge_to_pr_byteenable</td>
<td>4</td>
<td>Input</td>
<td>Optional Avalon-MM master bridge to PR region byteenable port.</td>
</tr>
<tr>
<td>mst_bridge_to_pr_writedata</td>
<td>32</td>
<td>Input</td>
<td>Optional Avalon-MM master bridge to PR region writedata port.</td>
</tr>
</tbody>
</table>

continued...
### Avalon Memory-Mapped Master to Static Region Slave Interface Ports

<table>
<thead>
<tr>
<th>Port</th>
<th>Width</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>mst_bridge_to_pr_readdata</code></td>
<td>32</td>
<td>Output</td>
<td>Optional Avalon memory-mapped master bridge to PR region <code>readdata</code> port.</td>
</tr>
<tr>
<td><code>mst_bridge_to_pr_burstcount</code></td>
<td>3</td>
<td>Input</td>
<td>Optional Avalon memory-mapped master bridge to PR region <code>burstcount</code> port.</td>
</tr>
<tr>
<td><code>mst_bridge_to_pr_readdatavalid</code></td>
<td>1</td>
<td>Output</td>
<td>Optional Avalon memory-mapped master bridge to PR region <code>readdatavalid</code> port.</td>
</tr>
<tr>
<td><code>mst_bridge_to_pr_beginbursttransfer</code></td>
<td>1</td>
<td>Input</td>
<td>Optional Avalon memory-mapped master bridge to PR region <code>beginbursttransfer</code> port.</td>
</tr>
<tr>
<td><code>mst_bridge_to_pr_debugaccess</code></td>
<td>1</td>
<td>Input</td>
<td>Optional Avalon memory-mapped master bridge to PR region <code>debugaccess</code> port.</td>
</tr>
<tr>
<td><code>mst_bridge_to_pr_response</code></td>
<td>2</td>
<td>Output</td>
<td>Optional Avalon memory-mapped master bridge to PR region <code>response</code> port.</td>
</tr>
<tr>
<td><code>mst_bridge_to_pr_lock</code></td>
<td>1</td>
<td>Input</td>
<td>Optional Avalon memory-mapped master bridge to PR region <code>lock</code> port.</td>
</tr>
<tr>
<td><code>mst_bridge_to_pr_writeresponsevalid</code></td>
<td>1</td>
<td>Output</td>
<td>Optional Avalon memory-mapped master bridge to PR region <code>writeresponsevalid</code> port.</td>
</tr>
</tbody>
</table>

Table 54. Avalon Memory-Mapped Master to Static Region Slave Interface Ports
Same setting as Avalon Memory-Mapped slave to PR region master interface.

<table>
<thead>
<tr>
<th>Port</th>
<th>Width</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>mst_bridge_to_sr_read</code></td>
<td>1</td>
<td>Output</td>
<td>Avalon memory-mapped master bridge to static region <code>read</code> port.</td>
</tr>
<tr>
<td><code>mst_bridge_to_sr_waitrequest</code></td>
<td>1</td>
<td>Input</td>
<td>Avalon memory-mapped bridge to static region <code>waitrequest</code> port.</td>
</tr>
<tr>
<td><code>mst_bridge_to_sr_write</code></td>
<td>1</td>
<td>Output</td>
<td>Avalon memory-mapped master bridge to static region <code>write</code> port.</td>
</tr>
<tr>
<td><code>mst_bridge_to_sr_address</code></td>
<td>32</td>
<td>Output</td>
<td>Avalon memory-mapped master bridge to static region <code>address</code> port.</td>
</tr>
<tr>
<td><code>mst_bridge_to_sr_byteenable</code></td>
<td>4</td>
<td>Output</td>
<td>Avalon memory-mapped master bridge to static region <code>byteenable</code> port.</td>
</tr>
<tr>
<td><code>mst_bridge_to_sr_writedata</code></td>
<td>32</td>
<td>Output</td>
<td>Avalon memory-mapped master bridge to static region <code>writedata</code> port.</td>
</tr>
<tr>
<td><code>mst_bridge_to_sr_readdata</code></td>
<td>32</td>
<td>Input</td>
<td>Avalon memory-mapped master bridge to static region <code>readdata</code> port.</td>
</tr>
<tr>
<td><code>mst_bridge_to_sr_burstcount</code></td>
<td>3</td>
<td>Output</td>
<td>Avalon memory-mapped master bridge to static region <code>burstcount</code> port.</td>
</tr>
<tr>
<td><code>mst_bridge_to_sr_readdatavalid</code></td>
<td>1</td>
<td>Input</td>
<td>Avalon memory-mapped master bridge to static region <code>readdatavalid</code> port.</td>
</tr>
<tr>
<td><code>mst_bridge_to_sr_beginbursttransfer</code></td>
<td>1</td>
<td>Output</td>
<td>Avalon memory-mapped master bridge to static region <code>beginbursttransfer</code> port.</td>
</tr>
<tr>
<td><code>mst_bridge_to_sr_debugaccess</code></td>
<td>1</td>
<td>Output</td>
<td>Avalon memory-mapped master bridge to static region <code>debugaccess</code> port.</td>
</tr>
</tbody>
</table>

*continued...*
### Avalon Memory-Mapped Master Interface Ports

<table>
<thead>
<tr>
<th>Port</th>
<th>Width</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mst_bridge_to_sr_response</td>
<td>2</td>
<td>Input</td>
<td>Avalon memory-mapped master bridge to static region response port.</td>
</tr>
<tr>
<td>mst_bridge_to_sr_lock</td>
<td>1</td>
<td>Output</td>
<td>Avalon memory-mapped master bridge to static region lock port.</td>
</tr>
<tr>
<td>mst_bridge_to_sr_writereqvalid</td>
<td>1</td>
<td>Input</td>
<td>Avalon memory-mapped master bridge to static region writereqvalid port.</td>
</tr>
</tbody>
</table>

#### Figure 69. Avalon Memory-Mapped Master Interface Ports

Slave Side of Bridge:

- `clock`
- `freeze_conduit`
- `freeze_conduit_freeze`
- `freeze_conduit_illegal_request`
- `mst_bridge_to_pr_read`
- `mst_bridge_to_pr_write`
- `mst_bridge_to_pr_address[31:0]`
- `mst_bridge_to_pr_writedata[31:0]`
- `mst_bridge_to_pr_burstcount[2:0]`
- `mst_bridge_to_pr_readvalid`
- `mst_bridge_to_pr_beginbursttransfer`
- `mst_bridge_to_pr_debugaccess`
- `mst_bridge_to_pr_responset[1:0]`
- `mst_bridge_to_pr_lock`
- `mst_bridge_to_sr_writereqvalid`
- `reset_n`

Master Side of Bridge:

- `clock`
- `read`
- `writereq`
- `write`
- `address`
- `byteenable`
- `writedata`
- `readdata`
- `burstcount`
- `readvalid`
- `beginbursttransfer`
- `debugaccess`
- `response`
- `lock`
- `writereqvalid`
- `reset_n`
2.7. Avalon Streaming Partial Reconfiguration Freeze Bridge IP

The Avalon Streaming Partial Reconfiguration Freeze Bridge Intel FPGA IP freezes a PR region Avalon streaming interface when the `freeze` input signal is high. The Avalon Streaming Partial Reconfiguration Freeze Bridge IP ensures that any transaction is complete before freezing the connected interface. It is recommended that each Avalon streaming interface to a PR region use an instance of the Freeze Bridge IP.
### Table 55. Avalon Streaming Source Freeze Bridge Interface Behavior

<table>
<thead>
<tr>
<th>Interface Type</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Source interface in the PR region with packet transfer (old or new persona)</strong></td>
<td>1. When the freeze signal goes high, the Freeze Bridge handles the startofpacket, endofpacket, and empty bits and does not send transactions to the static region.&lt;br&gt;2. When the Freeze Bridge detects a startofpacket transaction without a corresponding endofpacket during the frozen state, this indicates an unfinished transaction.&lt;br&gt;3. The bridge then completes the transaction by asserting valid and endofpacket high to the static region for one clock cycle.&lt;br&gt;4. The channel signal remains constant, while data bits are set to 'hDEADBEEF' and error bit is set to 1'b1.&lt;br&gt;5. The illegal_request output signal triggers update of the CSR register in the Partial Reconfiguration Region Controller.</td>
</tr>
<tr>
<td><strong>Source interface in the PR region without packet transfer (old or new persona)</strong></td>
<td>When the freeze signal is high, the Freeze Bridge does not send transactions to the static region. The Freeze Bridge remains idle until the bridge leaves the frozen state.</td>
</tr>
<tr>
<td><strong>Source interface in the PR region with max_channel &gt; 1 (old or new persona)</strong></td>
<td>When multiple channels transfer unfinished transactions, the Freeze Bridge tracks the channel values to ensure that all packet transactions from different channels end by asserting the endofpacket bit during the frozen state.</td>
</tr>
<tr>
<td><strong>Source interface in the PR region with ready_latency &gt; 0 (old or new persona)</strong></td>
<td>When the Freeze Bridge drives endofpacket, valid, or channel outputs to the static region, the Freeze Bridge reads the ready_latency value. The ready_latency value defines the actual clock cycle when the sink component is ready for data.</td>
</tr>
</tbody>
</table>

### Figure 72. Source Bridge Handling of Unfinished Packet Transaction During Freeze

![Diagram of clk, freeze, valid, data, ready, channel, error, endofpacket, startofpacket, empty, illegal_request signals during freeze state.](image-url)
Figure 73. PR Freeze Bridge Asserting valid Signal to End Packet Transactions

Table 56. Avalon Streaming Sink Freeze Bridge Interface Behavior

<table>
<thead>
<tr>
<th>Interface Type</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sink interface in PR region</td>
<td>For transactions that includes packet transfers, when the freeze signal goes high, the Freeze Bridge holds the ready signal high to the static region source until any unfinished transaction completes. For transactions that do not include packet transfers, when the freeze signal goes high, the Freeze Bridge holds the ready signal low during the freeze period. The illegal_request signal asserts high to indicate that the current transaction is an error. Configure the design to stop sending transactions to the PR region after the illegal_request signal is high.</td>
</tr>
<tr>
<td>Sink interface in PR region with ready_latency &gt; 0</td>
<td>When the Freeze Bridge drives endofpacket, valid, or channel outputs to the PR region, the Freeze Bridge must observe the ready_latency value. The ready_latency value defines the actual clock cycle when the sink component is ready for data.</td>
</tr>
</tbody>
</table>

2.7.1. Parameters

The Avalon Streaming Partial Reconfiguration Freeze Bridge IP core supports customization of the following parameters:
Figure 74. Parameter Editor

Table 57. Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PR region Interface Type</td>
<td>Avalon-ST Source/Avalon-ST Sink</td>
<td>Specifies the interface type for interfacing the PR region with the freeze bridge.</td>
</tr>
<tr>
<td>Enable Freeze port from PR region</td>
<td>On/Off</td>
<td>Enables the freeze port to freeze all the outputs of each PR region to a known constant value. Freezing prevents the signal receivers in the static region from receiving undefined signals during the partial reconfiguration process.</td>
</tr>
<tr>
<td>Select Yes or No to enable or disable interface ports</td>
<td>Yes/No</td>
<td>Enables or disables specific optional freeze bridge interface ports.</td>
</tr>
<tr>
<td>Channel width</td>
<td>&lt;1-128&gt;</td>
<td>Specifies the width of the channel signal.</td>
</tr>
<tr>
<td>Error width</td>
<td>&lt;1-256&gt;</td>
<td>Specifies the width of the error signal.</td>
</tr>
<tr>
<td>Data bits per symbol</td>
<td>&lt;1-512&gt;</td>
<td>Specifies the number of bits per symbol.</td>
</tr>
<tr>
<td>Symbols per beat</td>
<td>&lt;1-512&gt;</td>
<td>Specifies the number of symbols that transfer on every valid clock cycle.</td>
</tr>
</tbody>
</table>
### Parameter Table

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error descriptors</td>
<td>&lt;text&gt;</td>
<td>Specifies one or more strings to describe the error condition for each bit of the error port on the sink interface connected to the source interface. Click the plus or minus buttons to add or remove descriptors.</td>
</tr>
<tr>
<td>Max channel number</td>
<td>&lt;0-255&gt;</td>
<td>Specifies the maximum number of output channels.</td>
</tr>
<tr>
<td>Ready latency</td>
<td>&lt;0-8&gt;</td>
<td>Specifies what ready latency to expect from the source interface connected to the sink interface. The ready latency is the number of cycles from the time ready asserts until valid data is driven.</td>
</tr>
</tbody>
</table>

### 2.7.2.Ports

The Avalon Streaming Partial Reconfiguration Freeze Bridge IP has the following ports:

**Figure 75. Avalon Streaming Sink Interface Ports**

![Avalon Streaming Sink Interface Ports](image)
Figure 76. Avalon Streaming Source Interface Ports

<table>
<thead>
<tr>
<th>Port</th>
<th>Width</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clock</td>
<td>1</td>
<td>Input</td>
<td>Input clock for the IP.</td>
</tr>
<tr>
<td>freeze_conduit_freeze</td>
<td>1</td>
<td>Input</td>
<td>When this signal is high, the bridge handles any current transaction properly then freezes the PR interfaces.</td>
</tr>
<tr>
<td>freeze_conduit_illegal_request</td>
<td>1</td>
<td>Output</td>
<td>High on this bus indicates that an illegal request was issued to the bridge during the freeze state. $n$ = number of freeze bridge</td>
</tr>
<tr>
<td>pr_freeze_pr_freeze</td>
<td>1</td>
<td>Input</td>
<td>Enabled freeze port from the PR region.</td>
</tr>
<tr>
<td>reset_n</td>
<td>1</td>
<td>Input</td>
<td>Synchronous reset for the IP.</td>
</tr>
</tbody>
</table>

Table 59. Avalon Streaming Sink to Static Region Interface Ports

Same setting as Avalon streaming sink to PR region interface.

<table>
<thead>
<tr>
<th>Port</th>
<th>Width</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sink_bridge_to_sr_channel</td>
<td>1</td>
<td>Input</td>
<td>Avalon streaming sink bridge to static region channel port.</td>
</tr>
<tr>
<td>sink_bridge_to_sr_data</td>
<td>32</td>
<td>Input</td>
<td>Avalon streaming sink bridge to static region data port.</td>
</tr>
<tr>
<td>sink_bridge_to_sr_empty</td>
<td>2</td>
<td>Input</td>
<td>Avalon streaming sink bridge to static region empty port.</td>
</tr>
<tr>
<td>sink_bridge_to_sr_error</td>
<td>1</td>
<td>Input</td>
<td>Avalon streaming sink bridge to static region error port.</td>
</tr>
<tr>
<td>sink_bridge_to_sr_ready</td>
<td>1</td>
<td>Output</td>
<td>Avalon streaming sink bridge to static region ready port.</td>
</tr>
</tbody>
</table>

continued...
### Table 60. **Avalon-Streaming Sink to PR Region Interface Ports**

<table>
<thead>
<tr>
<th>Port</th>
<th>Width</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sink_bridge_to_pr_channel</td>
<td>1</td>
<td>Output</td>
<td>Optional Avalon streaming sink bridge to PR region channel port.</td>
</tr>
<tr>
<td>sink_bridge_to_pr_data</td>
<td>32</td>
<td>Output</td>
<td>Optional Avalon streaming sink bridge to PR region data port.</td>
</tr>
<tr>
<td>sink_bridge_to_pr_empty</td>
<td>2</td>
<td>Output</td>
<td>Optional Avalon streaming sink bridge to PR region empty port.</td>
</tr>
<tr>
<td>sink_bridge_to_pr_error</td>
<td>1</td>
<td>Output</td>
<td>Optional Avalon streaming sink bridge to PR region error port.</td>
</tr>
<tr>
<td>sink_bridge_to_pr_ready</td>
<td>1</td>
<td>Input</td>
<td>Optional Avalon-ST sink bridge to PR region ready port.</td>
</tr>
<tr>
<td>sink_bridge_to_pr_valid</td>
<td>1</td>
<td>Output</td>
<td>Optional Avalon streaming sink bridge to PR region valid port.</td>
</tr>
<tr>
<td>sink_bridge_to_pr_endofpacket</td>
<td>1</td>
<td>Output</td>
<td>Optional Avalon streaming sink bridge to PR region endofpacket port.</td>
</tr>
<tr>
<td>sink_bridge_to_pr_startofpacket</td>
<td>1</td>
<td>Output</td>
<td>Optional Avalon streaming sink bridge to PR region startofpacket port.</td>
</tr>
</tbody>
</table>

### Table 61. **Avalon Streaming Source to Static Region Interface Ports**

Same setting as Avalon streaming source to PR region interface.

<table>
<thead>
<tr>
<th>Port</th>
<th>Width</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source_bridge_to_sr_channel</td>
<td>1</td>
<td>Output</td>
<td>Avalon streaming source bridge to static region channel port.</td>
</tr>
<tr>
<td>source_bridge_to_sr_data</td>
<td>32</td>
<td>Output</td>
<td>Avalon streaming source bridge to static region data port.</td>
</tr>
<tr>
<td>source_bridge_to_sr_empty</td>
<td>2</td>
<td>Output</td>
<td>Avalon streaming source bridge to static region empty port.</td>
</tr>
<tr>
<td>source_bridge_to_sr_error</td>
<td>1</td>
<td>Output</td>
<td>Avalon streaming source bridge to static region error port.</td>
</tr>
<tr>
<td>source_bridge_to_sr_ready</td>
<td>1</td>
<td>Input</td>
<td>Avalon streaming source bridge to static region ready port.</td>
</tr>
<tr>
<td>source_bridge_to_sr_valid</td>
<td>1</td>
<td>Output</td>
<td>Avalon streaming source bridge to static region valid port.</td>
</tr>
<tr>
<td>source_bridge_to_sr_endofpacket</td>
<td>1</td>
<td>Output</td>
<td>Avalon streaming source bridge to static region endofpacket port.</td>
</tr>
<tr>
<td>source_bridge_to_sr_startofpacket</td>
<td>1</td>
<td>Output</td>
<td>Avalon streaming source bridge to static region startofpacket port.</td>
</tr>
</tbody>
</table>
Table 62. Avalon Streaming Source to PR Region Interface Ports

<table>
<thead>
<tr>
<th>Port</th>
<th>Width</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source_bridge_to_pr_channel</td>
<td>1</td>
<td>Input</td>
<td>Optional Avalon streaming source bridge to PR region channel port.</td>
</tr>
<tr>
<td>source_bridge_to_pr_data</td>
<td>32</td>
<td>Input</td>
<td>Optional Avalon streaming source bridge to PR region data port.</td>
</tr>
<tr>
<td>source_bridge_to_pr_empty</td>
<td>2</td>
<td>Input</td>
<td>Optional Avalon streaming source bridge to PR region empty port.</td>
</tr>
<tr>
<td>source_bridge_to_pr_error</td>
<td>1</td>
<td>Input</td>
<td>Optional Avalon-ST source bridge to PR region error port.</td>
</tr>
<tr>
<td>source_bridge_to_pr_ready</td>
<td>1</td>
<td>Output</td>
<td>Optional Avalon streaming source bridge to PR region ready port.</td>
</tr>
<tr>
<td>source_bridge_to_pr_valid</td>
<td>1</td>
<td>Input</td>
<td>Optional Avalon streaming source bridge to PR region valid port.</td>
</tr>
<tr>
<td>source_bridge_to_pr_endofpacket</td>
<td>1</td>
<td>Input</td>
<td>Optional Avalon streaming source bridge to PR region endofpacket port.</td>
</tr>
<tr>
<td>source_bridge_to_pr_startofpacket</td>
<td>1</td>
<td>Input</td>
<td>Optional Avalon streaming source bridge to PR region startofpacket port.</td>
</tr>
</tbody>
</table>

2.8. Generating and Simulating Intel FPGA IP

Use the following information to generate and simulate an IP core variation.

2.8.1. Generating IP Cores (Intel Quartus Prime Pro Edition)

Quickly configure Intel FPGA IP cores in the Intel Quartus Prime parameter editor. Double-click any component in the IP Catalog to launch the parameter editor. The parameter editor allows you to define a custom variation of the IP core. The parameter editor generates the IP variation synthesis and optional simulation files, and adds the .ip file representing the variation to your project automatically.

Follow these steps to locate, instantiate, and customize an IP core in the parameter editor:

1. Create or open an Intel Quartus Prime project (.qpf) to contain the instantiated IP variation.
2. In the IP Catalog (Tools ➤ IP Catalog), locate and double-click the name of the IP core to customize. To locate a specific component, type some or all of the component’s name in the IP Catalog search box. The New IP Variation window appears.
3. Specify a top-level name for your custom IP variation. Do not include spaces in IP variation names or paths. The parameter editor saves the IP variation settings in a file named <your_ip>.ip. Click OK. The parameter editor appears.
4. Set the parameter values in the parameter editor and view the block diagram for the component. The Parameterization Messages tab at the bottom displays any errors in IP parameters:
   - Optionally, select preset parameter values if provided for your IP core. Presets specify initial parameter values for specific applications.
   - Specify parameters defining the IP core functionality, port configurations, and device-specific features.
   - Specify options for processing the IP core files in other EDA tools.

   Note: Refer to your IP core user guide for information about specific IP core parameters.

5. Click Generate HDL. The Generation dialog box appears.

6. Specify output file generation options, and then click Generate. The synthesis and simulation files generate according to your specifications.

7. To generate a simulation testbench, click Generate ➤ Generate Testbench System. Specify testbench generation options, and then click Generate.

8. To generate an HDL instantiation template that you can copy and paste into your text editor, click Generate ➤ Show Instantiation Template.

9. Click Finish. Click Yes if prompted to add files representing the IP variation to your project.

10. After generating and instantiating your IP variation, make appropriate pin assignments to connect ports.
Some IP cores generate different HDL implementations according to the IP core parameters. The underlying RTL of these IP cores contains a unique hash code that prevents module name collisions between different variations of the IP core. This unique code remains consistent, given the same IP settings and software version during IP generation. This unique code can change if you edit the IP core's parameters or upgrade the IP core version. To avoid dependency on these unique codes in your simulation environment, refer to *Generating a Combined Simulator Setup Script*.

**Related Information**
- Introduction to Intel FPGA IP Cores
- Generating a Combined Simulator Setup Script

### 2.8.2. Running the Freeze Bridge Update script

When instantiating the Freeze Bridge as a Platform Designer system component, the interface connections between the Freeze Bridge and the PR region must match, so that Platform Designer inserts no extra interconnect during system generation. Rather than manually matching the Avalon interface properties individually in the parameter editor, you can run the provided Update Freeze Bridge Parameters script to update Freeze Bridge Avalon interface properties automatically.

Running this script updates the master and slave interfaces or the sink and source interfaces of the Freeze Bridge, according to the Avalon property settings of the connecting PR region component.

**To run the Update Freeze Bridge Parameters script:**

1. Open a Platform Designer system containing one or more instances of the Freeze Bridge component.
2. In Platform Designer, click **View ➤ System Scripting**. The **System Scripting** tab displays **Platform Designer Built-in Scripts**.
3. To update all freeze bridges in your Platform Designer system, set `update_all_freeze_bridges` to 1 in the Additional Commands section of the script. To update only a single freeze bridge, click the freeze bridge instance.
4. Click **Run Script**. The script runs and updates the freeze bridge parameters.
2.8.3. IP Core Generation Output (Intel Quartus Prime Pro Edition)

The Intel Quartus Prime software generates the following output file structure for individual IP cores that are not part of a Platform Designer system.
Figure 79. Individual IP Core Generation Output (Intel Quartus Prime Pro Edition)

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;Project Directory&gt;</code></td>
</tr>
<tr>
<td><code>&lt;your_ip&gt;.ip</code> - Top-level IP variation file</td>
</tr>
<tr>
<td><code>&lt;your_ip&gt;</code> - IP core variation files</td>
</tr>
<tr>
<td><code>&lt;your_ip&gt;.bsf</code> - Block symbol schematic file</td>
</tr>
<tr>
<td><code>&lt;your_ip&gt;.cmp</code> - VHDL component declaration</td>
</tr>
<tr>
<td><code>&lt;your_ip&gt;.ppf</code> - XML I/O pin information file</td>
</tr>
<tr>
<td><code>&lt;your_ip&gt;.qip</code> - Lists files for IP core synthesis</td>
</tr>
<tr>
<td><code>&lt;your_ip&gt;.spd</code> - Simulation startup scripts</td>
</tr>
<tr>
<td><code>&lt;your_ip&gt;_bb.v</code> - Verilog HDL black box EDA synthesis file *</td>
</tr>
<tr>
<td><code>&lt;your_ip&gt;_generation.rpt</code> - IP generation report</td>
</tr>
<tr>
<td><code>&lt;your_ip&gt;_inst.v</code> or <code>.vhdl</code> - Lists file for IP core synthesis</td>
</tr>
<tr>
<td><code>&lt;your_ip&gt;.qgsimc</code> - Simulation caching file (Platform Designer)</td>
</tr>
<tr>
<td><code>&lt;your_ip&gt;.qgsynthc</code> - Synthesis caching file (Platform Designer)</td>
</tr>
<tr>
<td>sim - IP simulation files</td>
</tr>
<tr>
<td><code>&lt;your_ip&gt;.v</code> or <code>.vhdl</code> - Top-level simulation file</td>
</tr>
<tr>
<td><code>&lt;simulator vendor&gt;</code> - Simulator setup scripts</td>
</tr>
<tr>
<td><code>&lt;simulator_setup_scripts&gt;</code></td>
</tr>
<tr>
<td>synth - IP synthesis files</td>
</tr>
<tr>
<td><code>&lt;your_ip&gt;.v</code> or <code>.vhdl</code> - Top-level IP synthesis file</td>
</tr>
<tr>
<td><code>&lt;IP Submodule&gt;_&lt;version&gt;</code> - IP Submodule Library</td>
</tr>
<tr>
<td>sim - IP submodule 1 simulation files</td>
</tr>
<tr>
<td><code>&lt;HDL files&gt;</code></td>
</tr>
<tr>
<td>synth - IP submodule 1 synthesis files</td>
</tr>
<tr>
<td><code>&lt;HDL files&gt;</code></td>
</tr>
<tr>
<td><code>&lt;your_ip&gt;_tb</code> - IP testbench system *</td>
</tr>
<tr>
<td><code>&lt;your_testbench&gt;_tb.qsys</code> - testbench system file</td>
</tr>
<tr>
<td><code>&lt;your_ip&gt;_tb</code> - IP testbench files</td>
</tr>
<tr>
<td><code>&lt;your_testbench&gt;_tb.csv</code> or <code>.spd</code> - testbench file</td>
</tr>
<tr>
<td>sim - IP testbench simulation files</td>
</tr>
</tbody>
</table>

* If supported and enabled for your IP core variation.

Table 63. Output Files of Intel FPGA IP Generation

<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;your_ip&gt;.ip</code></td>
<td>Top-level IP variation file that contains the parameterization of an IP core in your project. If the IP variation is part of a Platform Designer system, the parameter editor also generates a <code>.qsys</code> file.</td>
</tr>
<tr>
<td><code>&lt;your_ip&gt;.cmp</code></td>
<td>The VHDL Component Declaration (<code>.cmp</code>) file is a text file that contains local generic and port definitions that you use in VHDL design files.</td>
</tr>
<tr>
<td><code>&lt;your_ip&gt;_generation.rpt</code></td>
<td>IP or Platform Designer generation log file. Displays a summary of the messages during IP generation.</td>
</tr>
</tbody>
</table>

continued...
<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;your_ip&gt;.qgsimc</code> (Platform Designer systems only)</td>
<td>Simulation caching file that compares the .qsys and .ip files with the current parameterization of the Platform Designer system and IP core. This comparison determines if Platform Designer can skip regeneration of the HDL.</td>
</tr>
<tr>
<td><code>&lt;your_ip&gt;.qgsynth</code> (Platform Designer systems only)</td>
<td>Synthesis caching file that compares the .qsys and .ip files with the current parameterization of the Platform Designer system and IP core. This comparison determines if Platform Designer can skip regeneration of the HDL.</td>
</tr>
<tr>
<td><code>&lt;your_ip&gt;.csv</code></td>
<td>Contains information about the upgrade status of the IP component.</td>
</tr>
<tr>
<td><code>&lt;your_ip&gt;.bsf</code></td>
<td>A symbol representation of the IP variation for use in Block Diagram Files (.bdf).</td>
</tr>
<tr>
<td><code>&lt;your_ip&gt;.spd</code></td>
<td>Input file that <code>ip-make-simscript</code> requires to generate simulation scripts. The .spd file contains a list of files you generate for simulation, along with information about memories that you initialize.</td>
</tr>
<tr>
<td><code>&lt;your_ip&gt;.ppf</code></td>
<td>The Pin Planner File (.ppf) stores the port and node assignments for IP components you create for use with the Pin Planner.</td>
</tr>
<tr>
<td><code>&lt;your_ip&gt;_.bb.v</code></td>
<td>Use the Verilog blackbox (<code>_.bb.v</code>) file as an empty module declaration for use as a blackbox.</td>
</tr>
<tr>
<td><code>&lt;your_ip&gt;_.inst.v</code> or <code>.inst.vhd</code></td>
<td>HDL example instantiation template. Copy and paste the contents of this file into your HDL file to instantiate the IP variation.</td>
</tr>
<tr>
<td><code>&lt;your_ip&gt;_.regmap</code></td>
<td>If the IP contains register information, the Intel Quartus Prime software generates the .regmap file. The .regmap file describes the register map information of master and slave interfaces. This file complements the .sopcinfo file by providing more detailed register information about the system. This file enables register display views and user customizable statistics in System Console.</td>
</tr>
<tr>
<td><code>&lt;your_ip&gt;_.svd</code></td>
<td>Allows HPS System Debug tools to view the register maps of peripherals that connect to HPS within a Platform Designer system. During synthesis, the Intel Quartus Prime software stores the .svd files for slave interface visible to the System Console masters in the .sof file in the debug session. System Console reads this section, which Platform Designer queries for register map information. For system slaves, Platform Designer accesses the registers by name.</td>
</tr>
<tr>
<td><code>&lt;your_ip&gt;_.v</code> <code>&lt;your_ip&gt;_.vhdx</code></td>
<td>HDL files that instantiate each submodule or child IP core for synthesis or simulation.</td>
</tr>
<tr>
<td>mentor/</td>
<td>Contains a <code>msim_setup.tcl</code> script to set up and run a ModelSim* simulation.</td>
</tr>
<tr>
<td>aldec/</td>
<td>Contains a Riviera-PRO* script <code>rivierapro_setup.tcl</code> to set up and run a simulation.</td>
</tr>
<tr>
<td>/synopsys/vcs</td>
<td>Contains a shell script <code>vcs_setup.sh</code> to set up and run a VCS* simulation.</td>
</tr>
<tr>
<td>/synopsys/vcsmx</td>
<td>Contains a shell script <code>vcsmx_setup.sh</code> and <code>synopsys_sim.setup</code> file to set up and run a VCS MX simulation.</td>
</tr>
<tr>
<td>/cadence</td>
<td>Contains a shell script <code>ncsim_setup.sh</code> and other setup files to set up and run an NCsim simulation.</td>
</tr>
<tr>
<td>/xcelium</td>
<td>Contains an Xcelium* Parallel simulator shell script <code>xcelium_setup.sh</code> and other setup files to set up and run a simulation.</td>
</tr>
<tr>
<td>/submodules</td>
<td>Contains HDL files for the IP core submodule.</td>
</tr>
<tr>
<td><code>&lt;IP submodule&gt;/</code></td>
<td>Platform Designer generates <code>/synth</code> and <code>/sim</code> sub-directories for each IP submodule directory that Platform Designer generates.</td>
</tr>
</tbody>
</table>
2.8.4. Intel Arria 10 and Intel Cyclone 10 GX PR Control Block Simulation Model

The Intel Quartus Prime Pro Edition software supports simulating the delivery of a partial reconfiguration bitstream to the PR control block. This simulation allows you to observe the resulting change and the intermediate effect in a reconfigurable partition.

The Intel Arria 10 and Intel Cyclone 10 GX PR control blocks support PR simulation. Sending a simulation RBF (PR bitstream) allows the PR control block to behave accordingly, to PR simulation success or PR simulation failure. To activate simulation of a specific PR persona in your PR region simulation wrapper, use a PR ID encoded in the simulation RBF, in conjunction with the PR control block. Simulate the PR control block either as standalone, or as part of the simulation file set for the Partial Reconfiguration Controller IP core.

Figure 80. PR Control Block Simulation Model

The PR control block simulation model contains two additional simulation-only ports—sim_state and sim_pr_id. Connect these simulation ports, and the other ports, to the twentynm_prblock_if SystemVerilog interface. This connection allows monitoring of the PR control block using your testbench’s PR control block monitor. The Intel Quartus Prime software automatically instantiates the twentynm_prblock_if interface when generating the simulation file set of the Partial Reconfiguration IP core. Obtain a reference to the twentynm_prblock_if that the IP instantiates by using the alt_pr_test_pkg::twentynm_prblock_if_mgr singleton, as shown in the following example:

```verilog
virtual twentynm_prblock_if prblock_if;
alt_pr_test_pkg::twentynm_prblock_if_mgr cb_mgr;

// Get the PR control block from the prblock manager
cb_mgr = alt_pr_test_pkg::twentynm_prblock_if_mgr::get();
prblock_if = cb_mgr.if_ref;
```

The code for the twentynm_prblock_if interface is as follows:

```verilog
interface twentynm_prblock_if(input logic pr_clk, input logic clk);

logic prrequest;
logic [31:0] data;
wire error;
wire ready;
wire done;
logic [31:0] sim_only_state;
wire [31:0] sim_only_pr_id;

// All signals are async except data
```
clocking cb1 @(posedge pr_clk);
  output data;
endclocking
endinterface : twentynm_prblock_if

For more information on the `twentynm_prblock_if` interface, refer to the `<installation directory>/eda/sim_lib/altera_lnsim.sv` file.

The simulation state of the PR control block simulation model represents the `PR_EVENT_TYPE` enumeration state of the control block. The `twentynm_prblock_test_pkg` SystemVerilog package defines these enumerations. These states represent the different allowed states for the control block. The defined control block enumerations are:

```
package twentynm_prblock_test_pkg;
typedef enum logic [31:0] {
  NONE,
  IDLE,
  PR_REQUEST,
  PR_IN_PROGRESS,
  PR_COMPLETE_SUCCESS,
  PR_COMPLETE_ERROR,
  PR_INCOMPLETE_EARLY_WITHDRAWL,
  PR_INCOMPLETE_LATE_WITHDRAWL
} PR_EVENT_TYPE;
```

When the simulation state is `PR_IN_PROGRESS`, the affected PR region must have its simulation output multiplexes driven to X, by asserting the `pr_activate` signal. This action simulates the unknown outputs of the PR region during partial reconfiguration. In addition, you must assert the `pr_activate` signal in the PR simulation model to load all registers in the PR model with the PR activation value.

Once the simulation state reaches `PR_COMPLETE_SUCCESS`, activate the appropriate PR persona using the appropriate PR region simulation wrapper `mux sel` signals. You can decode the region, as well as the specific select signal from the `sim_only_pr_id` signal of the PR control block. This ID corresponds to the encoded ID in the simulation RBF.

**Table 64. Required Sequence of Words in Simulation RBF**

<table>
<thead>
<tr>
<th>Step</th>
<th>Word</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>zero padding blocks</td>
<td>0x00000000</td>
</tr>
<tr>
<td>2</td>
<td>PR_HEADER_WORD</td>
<td>0x0000A65C</td>
</tr>
<tr>
<td>3</td>
<td>PR_ID</td>
<td>32-bit user ID</td>
</tr>
<tr>
<td>4</td>
<td>PRDATA_COUNT_0</td>
<td>0x01234567</td>
</tr>
<tr>
<td>5</td>
<td>PRDATA_COUNT_1</td>
<td>0x89ABCDEF</td>
</tr>
<tr>
<td>6</td>
<td>PRDATA_COUNT_2</td>
<td>0x02468ACE</td>
</tr>
<tr>
<td>7</td>
<td>PRDATA_COUNT_3</td>
<td>0x13579BDF</td>
</tr>
</tbody>
</table>

**Note:** The PR_ID word is output on the `sim_only_pr_id` word, starting at `PRDATA_COUNT_0`. Using a different value for the header or data count results in PR simulation errors.
Related Information

- Simulating PR Persona Replacement on page 50
- Simulating Intel FPGA IP Cores

2.8.5. Generating the PR Persona Simulation Model

Use the Intel Quartus Prime EDA Netlist Writer to create the simulation model for a PR persona. The simulation model represents the post-synthesis, gate-level netlist for the persona.

When using the PR simulation model for the persona, the netlist includes a new `altera_sim_pr_activate` top-level signal for the model. You can asynchronously drive this signal to load all registers in the model with X. This feature allows you to verify the reset sequence of the new persona on PR event completion. Verify the reset sequence through inspection, using SystemVerilog assertions, or using other checkers.

By default, the PR simulation model asynchronously loads X into the register’s storage element on `pr_activate` signal assertion. You can parameterize this behavior on a per register basis, or on a simulation-wide default basis. The simulation model supports four built-in modes:

- load X
- load 1
- load 0
- load rand

Specify these modes using the SystemVerilog classes:

- `dffeas_pr_load_x`
- `dffeas_load_1`
- `dffeas_load_0`
- `dffeas_load_rand`

Optionally, you can create your own PR activation class, where your class must define the `pr_load` variable to specify the PR activation value.

Follow these steps to generate the simulation model for a PR design:

1. Open the base revision of a PR project in Intel Quartus Prime Pro Edition, and then click **Processing ➤ Start ➤ Start Analysis & Synthesis**. Alternatively, run this command-line equivalent:

   ```bash
   quartus_syn <project name> -c <base revision name>
   ```

2. After synthesis is complete, click **Project ➤ Export Design Partition**, and then select the **root partition** for the **Partition name**, and select **synthesized** for the **Snapshot**. Click OK. Alternatively, run this command-line equivalent:

   ```bash
   quartus_cdb <project name> -c <base revision name> \
   --export_block root_partition --snapshot synthesized \
   --file <static qdb name>
   ```

3. Click **Project ➤ Revisions** and switch the current revision to that of the persona you want to export.
4. Click Processing ➤ Start ➤ Start Analysis & Synthesis. Alternatively, run this command-line equivalent:

```
quartus_syn <project name> -c <persona revision name>
```

5. After synthesis of the persona revision completes, execute the following at the command line to generate the PR simulation model:

```
quartus_eda <project name> -c <persona revision name> *--pr --simulation \ 
--tool=modelsim --format=verilog --partition=<pr partition name> \ 
--module=<persona module name>
```

6. Repeat steps 3 through 5 for all personas that you want to simulate.

**Example 6. Complete PR Simulation Model Generation Script**

```
quartus_syn <project name> -c <base revision name>
quartus_cdb <project name> -c <base revision name> \ 
  "--export_block root_partition --snapshot synthesized \ 
  --file <static qdb name>
quartus_syn <project name> -c <persona revision name>
quartus_eda <project name> -c <persona revision name> \ 
  "--pr --simulation --tool=modelsim --format=verilog \ 
  --partition=<pr partition name> --module=<persona module name>"
```

You can use the PR mode of the EDA netlist writer to generate the gate level netlist of a PR region. Refer to the "EDA Netlist Writer and Gate Level-Netlists" section of the Intel Quartus Prime Pro Edition User Guide: Third Party Simulation.

**Related Information**


If the table does not list a software version, the user guide for the previous software version applies.

<table>
<thead>
<tr>
<th>Intel Quartus Prime Version</th>
<th>User Guide</th>
</tr>
</thead>
<tbody>
<tr>
<td>18.0</td>
<td>Intel Quartus Prime Pro Edition User Guide: Partial Reconfiguration</td>
</tr>
</tbody>
</table>
## 2.10. Partial Reconfiguration Solutions IP User Guide Revision History

<table>
<thead>
<tr>
<th>Document Version</th>
<th>Intel Quartus Prime Version</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>2020.09.28</td>
<td>20.3</td>
<td>• Replaced references to Avalon-MM and Avalon-ST with Avalon memory-mapped and Avalon streaming for legal compliance.</td>
</tr>
<tr>
<td>2020.08.07</td>
<td>20.2</td>
<td>• Corrected signal name typos in &quot;Interface Ports&quot; topic for Avalon-MM Partial Reconfiguration Freeze Bridge Intel FPGA IP.</td>
</tr>
<tr>
<td>2019.12.16</td>
<td>19.4.0</td>
<td>• Added &quot;Error Detection CRC Requirements&quot; topic.</td>
</tr>
<tr>
<td>2019.09.30</td>
<td>19.3.0</td>
<td>• Updated the name of &quot;Intel Stratix 10 Partial Reconfiguration Controller FPGA IP&quot; to &quot;Partial Reconfiguration Controller Intel FPGA IP&quot; to encompass support for Intel Agilex devices. • Added note about connection of dummy_clk in &quot;PR Control Block and CRC Block VHDL Module.&quot; • Added note to &quot;PR Bitstream Compression and Encryption&quot; topic about support for enhanced decompression.</td>
</tr>
<tr>
<td>2019.06.07</td>
<td>19.1.0</td>
<td>• Added note and reference to Intel Stratix 10 Configuration User Guide.</td>
</tr>
<tr>
<td>2019.04.22</td>
<td>19.1.0</td>
<td>• Indicated support for POF generation support for Intel Cyclone GX devices.</td>
</tr>
<tr>
<td>2019.01.04</td>
<td>18.1.0</td>
<td>• Clarified statement about configuration width in &quot;Control Block Signals&quot; topic.</td>
</tr>
<tr>
<td>2018.12.07</td>
<td>18.1.0</td>
<td>• Corrected typographical error in &quot;Partial Reconfiguration IP Cores&quot; table. • Corrected typographical error in &quot;Avalon-MM Slave to PR Region Master Interface Ports&quot; table.</td>
</tr>
<tr>
<td>2018.09.24</td>
<td>18.1.0</td>
<td>• Updated specification for Partial Reconfiguration Controller Intel Stratix 10 FPGA IP from 250 MHz to 200 MHz. • Stated PR compilation flow support for Intel Cyclone 10 GX devices. • Updated Partial Reconfiguration Controller Intel Arria 10 FPGA IP name to Partial Reconfiguration Controller Intel Arria 10/Cyclone 10 FPGA IP.</td>
</tr>
<tr>
<td>2018.06.27</td>
<td>18.0.0</td>
<td>• Updated freeze_status signal description in Registers: Partial Reconfiguration Region Controller.</td>
</tr>
<tr>
<td>2018.06.18</td>
<td>18.0.0</td>
<td>• Corrected syntax error in Generating the PR Persona Simulation Model.</td>
</tr>
<tr>
<td>2018.05.07</td>
<td>18.0.0</td>
<td>• Added description of new Partial Reconfiguration External Configuration Controller Intel Stratix 10 FPGA IP. • Updated names of Partial Reconfiguration Controller Intel Arria 10 FPGA IP and Partial Reconfiguration Controller Intel Stratix 10 FPGA IP. • Enhanced explanation of Auto-instantiate CRC block Partial Reconfiguration Controller Intel Arria 10 parameter. • Added as chapter in new Partial Reconfiguration User Guide. • Added note about recovery after PR error when using SEU detection in Intel Stratix 10 designs.</td>
</tr>
<tr>
<td>2017.11.06</td>
<td>17.1.0</td>
<td>• Added support for Intel Stratix 10 Partial Reconfiguration Controller IP core. • Updated for latest Intel product naming conventions.</td>
</tr>
<tr>
<td>2017.05.08</td>
<td>17.0.0</td>
<td>Initial public release.</td>
</tr>
</tbody>
</table>

A. Intel Quartus Prime Pro Edition User Guides

Refer to the following user guides for comprehensive information on all phases of the Intel Quartus Prime Pro Edition FPGA design flow.

Related Information

  Introduces the basic features, files, and design flow of the Intel Quartus Prime Pro Edition software, including managing Intel Quartus Prime Pro Edition projects and IP, initial design planning considerations, and project migration from previous software versions.

  Describes creating and optimizing systems using Platform Designer, a system integration tool that simplifies integrating customized IP cores in your project. Platform Designer automatically generates interconnect logic to connect intellectual property (IP) functions and subsystems.

  Describes best design practices for designing FPGAs with the Intel Quartus Prime Pro Edition software. HDL coding styles and synchronous design practices can significantly impact design performance. Following recommended HDL coding styles ensures that Intel Quartus Prime Pro Edition synthesis optimally implements your design in hardware.

  Describes set up, running, and optimization for all stages of the Intel Quartus Prime Pro Edition Compiler. The Compiler synthesizes, places, and routes your design before generating a device programming file.

  Describes Intel Quartus Prime Pro Edition settings, tools, and techniques that you can use to achieve the highest design performance in Intel FPGAs. Techniques include optimizing the design netlist, addressing critical chains that limit retiming and timing closure, optimizing device resource usage, device floorplanning, and implementing engineering change orders (ECOs).

  Describes operation of the Intel Quartus Prime Pro Edition Programmer, which allows you to configure Intel FPGA devices, and program CPLD and configuration devices, via connection with an Intel FPGA download cable.

- Intel Quartus Prime Pro Edition User Guide: Block-Based Design
  Describes block-based design flows, also known as modular or hierarchical design flows. These advanced flows enable preservation of design blocks (or logic that comprises a hierarchical design instance) within a project, and reuse of design blocks in other projects.
  Describes Partial Reconfiguration, an advanced design flow that allows you to reconfigure a portion of the FPGA dynamically, while the remaining FPGA design continues to function. Define multiple personas for a particular design region, without impacting operation in other areas.

- **Intel Quartus Prime Pro Edition User Guide: Third-party Simulation**
  Describes RTL- and gate-level design simulation support for third-party simulation tools by Aldec*, Cadence*, Mentor Graphics*, and Synopsys* that allow you to verify design behavior before device programming. Includes simulator support, simulation flows, and simulating Intel FPGA IP.

  Describes support for optional synthesis of your design in third-party synthesis tools by Mentor Graphics*, and Synopsys*. Includes design flow steps, generated file descriptions, and synthesis guidelines.

  Describes support for optional logic equivalence checking (LEC) of your design in third-party LEC tools by OneSpin*.

  Describes a portfolio of Intel Quartus Prime Pro Edition in-system design debugging tools for real-time verification of your design. These tools provide visibility by routing (or "tapping") signals in your design to debugging logic. These tools include System Console, Signal Tap logic analyzer, system debugging toolkits, In-System Memory Content Editor, and In-System Sources and Probes Editor.

  Explains basic static timing analysis principals and use of the Intel Quartus Prime Pro Edition Timing Analyzer, a powerful ASIC-style timing analysis tool that validates the timing performance of all logic in your design using an industry-standard constraint, analysis, and reporting methodology.

  Describes the Intel Quartus Prime Pro Edition Power Analysis tools that allow accurate estimation of device power consumption. Estimate the power consumption of a device to develop power budgets and design power supplies, voltage regulators, heat sink, and cooling systems.

- **Intel Quartus Prime Pro Edition User Guide: Design Constraints**
  Describes timing and logic constraints that influence how the Compiler implements your design, such as pin assignments, device options, logic options, and timing constraints. Use the Interface Planner to prototype interface implementations, plan clocks, and quickly define a legal device floorplan. Use the Pin Planner to visualize, modify, and validate all I/O assignments in a graphical representation of the target device.

  Describes support for optional third-party PCB design tools by Mentor Graphics* and Cadence*. Also includes information about signal integrity analysis and simulations with HSPICE and IBIS Models.

- **Intel Quartus Prime Pro Edition User Guide: Scripting**
  Describes use of Tcl and command line scripts to control the Intel Quartus Prime Pro Edition software and to perform a wide range of functions, such as managing projects, specifying constraints, running compilation or timing analysis, or generating reports.