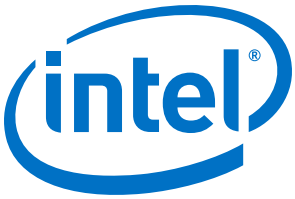


Intel® Quartus® Prime Standard Edition User Guide

Partial Reconfiguration

Updated for Intel® Quartus® Prime Design Suite: **18.1**

Subscribe
Send Feedback

UG-20179 | 2018.09.24
Latest document on the web: PDF | HTML

Contents

1. Design Planning for Partial Reconfiguration

The Partial Reconfiguration (PR) feature in the Intel® Quartus® Prime software allows you to reconfigure a portion of the FPGA dynamically, while the remainder of the device continues to operate.

This chapter assumes a basic knowledge of Altera's FPGA design flow, incremental compilation, and LogicLock™ region features available in the Intel Quartus Prime software. It also assumes knowledge of the internal FPGA resources such as logic array blocks (LABs), memory logic array blocks (MLABs), memory types (RAM and ROM), DSP blocks, clock networks.

The Intel Quartus Prime software supports the PR feature for the Intel Stratix® V device family and Cyclone® V devices whose part number ends in "SC", for example, 5CGXFC9E6F35I8NSC.

Related Information

1.1. Terminology

The following terms are commonly used in this chapter.

- **project:** A Intel Quartus Prime project contains the design files, settings, and constraints files required for the compilation of your design.
- **revision:** In the Intel Quartus Prime software, a revision is a set of assignments and settings for one version of your design. A Intel Quartus Prime project can have several revisions, and each revision has its own set of assignments and settings. A revision helps you to organize several versions of your design into a single project.
- **incremental compilation:** This is a feature of the Intel Quartus Prime software that allows you to preserve results of previous compilations of unchanged parts of the design, while changing the implementation of the parts of your design that you have modified since your previous compilation of the project. The key benefits include timing preservation and compile time reduction by only compiling the logic that has changed.
- **partition:** You can partition your design along logical hierarchical boundaries. Each design partition is independently synthesized and then merged into a complete netlist for further stages of compilation. With the Intel Quartus Prime incremental compilation flow, you can preserve results of unchanged partitions at specific preservation levels. For example, you can set the preservation levels at post-synthesis or post-fit, for iterative compilations in which some part of the design is changed. A partition is only a logical partition of the design, and does not necessarily refer to a physical location on the device. However, you may associate a partition with a specific area of the FPGA by using a floorplan assignment.

  For more information on design partitions, refer to the *Best Practices for Incremental Compilation Partitions and Floorplan Assignments* chapter in the *Intel Quartus Prime Handbook*.
- **LogicLock region:** A LogicLock region constrains the placement of logic in your design. You can associate a design partition with a LogicLock region to constrain the placement of the logic in the partition to a specific physical area of the FPGA.

  For more information about LogicLock regions, refer to the *Analyzing and Optimizing the Design Floorplan* chapter in the *Intel Quartus Prime Handbook* Volume 2.
- **PR project:** Any Intel Quartus Prime design project that uses the PR feature.
- **PR region:** A design partition with an associated contiguous LogicLock region in a PR project. A PR project can have one or more PR regions that can be partially reconfigured independently. A PR region may also be referred to as a PR partition.
- **static region:** The region outside of all the PR regions in a PR project that cannot be reprogrammed with partial reconfiguration (unless you reprogram the entire FPGA). This region is called the static region, or fixed region.
- **persona:** A PR region has multiple implementations. Each implementation is called a persona. PR regions can have multiple personas. In contrast, static regions have a single implementation or persona.
- **PR control block:** Dedicated block in the FPGA that processes the PR requests, handshake protocols, and verifies the CRC.
- **PR IP Core:** Altera soft IP that can be used to configure the PR control block in the FPGA to mange the PR bitstream source.

Related Information

Analyzing and Optimizing the Design Floorplan

1.1.1. Determining Resources for Partial Reconfiguration

You can use partial reconfiguration to configure only the resources such as LABs, embedded memory blocks, and DSP blocks in the FPGA core fabric that are controlled by configuration RAM (CRAM).

The functions in the periphery, such as GPIOs or I/O Registers, are controlled by I/O configuration bits and therefore cannot be partially reconfigured. Clock multiplexers for GCLK and QCLK are also not partially reconfigurable because they are controlled by I/O periphery bits.

Figure 1. Partially Reconfigurable Resources

These are the types of resource blocks in a Stratix V device.

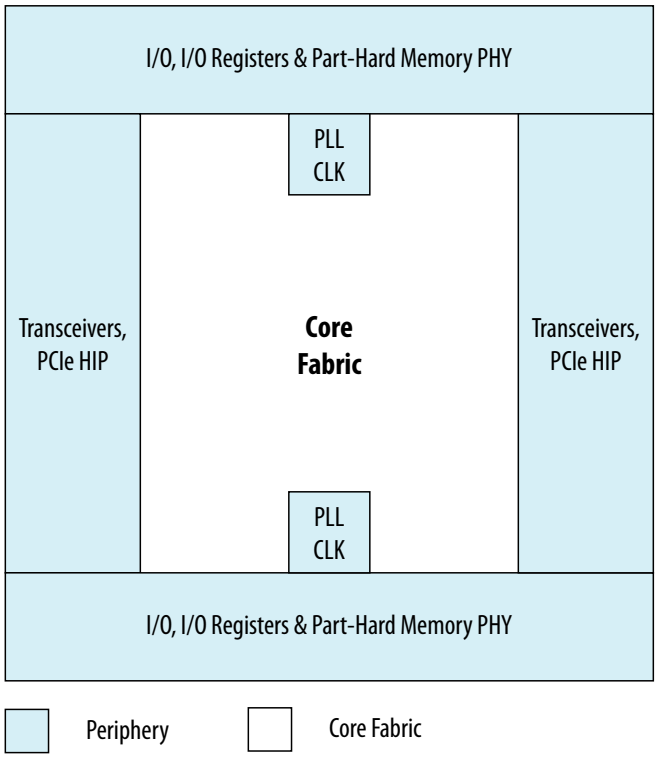


Table 1. Reconfiguration Modes of the FPGA Resource Block

The following table describes the reconfiguration type supported by each FPGA resource block, which are shown in the figure.

Hardware Resource Block	Reconfiguration Mode
Logic Block	Partial Reconfiguration
Digital Signal Processing	Partial Reconfiguration
Memory Block	Partial Reconfiguration
Transceivers	Dynamic Reconfiguration ALTGX_Reconfig
PLL	Dynamic Reconfiguration ALTGX_Reconfig

continued...

Hardware Resource Block	Reconfiguration Mode
Core Routing	Partial Reconfiguration
Clock Networks	Clock network sources cannot be changed, but a PLL driving a clock network can be dynamically reconfigured
I/O Blocks and Other Periphery	Not supported

The transceivers and PLLs in Altera FPGAs can be reconfigured using dynamic reconfiguration. For more information on dynamic reconfiguration, refer to the *Dynamic Reconfiguration in Stratix V Devices* chapter in the *Stratix V Handbook*.

Related Information

Dynamic Reconfiguration in Stratix V Devices

1.2. An Example of a Partial Reconfiguration Design

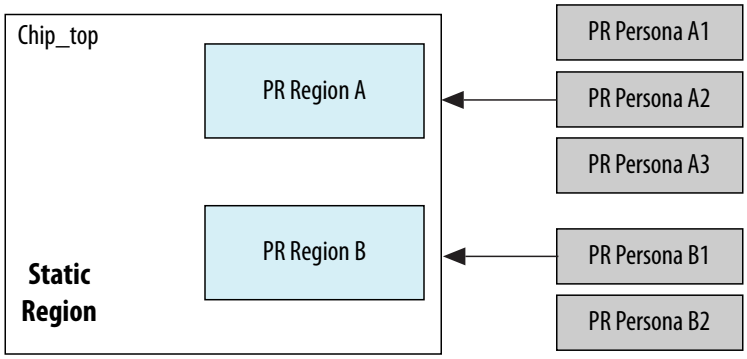
A PR design is divided into two parts. The static region where the design logic does not change, and one or more PR regions.

Each PR region can have different design personas, that change with partial reconfiguration.

PR Region A has three personas associated with it; A1, A2, and A3. PR Region B has two personas; B1 and B2. Each persona for the two PR regions can implement different application specific logic, and using partial reconfiguration, the persona for each PR region can be modified without interrupting the operation of the device in the static or other PR region.

Figure 2. Partial Reconfiguration Project Structure

The following figure shows the top-level of a PR design, which includes a static region and two PR regions.



1.3. Partial Reconfiguration Modes

When you implement a design on an Altera FPGA device, your design implementation is controlled by bits stored in CRAM inside the FPGA.

You can use partial reconfiguration in the SCRUB mode or the AND/OR mode. The mode you select affects your PR flow in ways detailed later in this chapter.

The CRAM bits control individual LABs, MLABs, M20K memory blocks, DSP blocks, and routing multiplexers in a design. The CRAM bits are organized into a frame structure representing vertical areas that correspond to specific locations on the FPGA. If you change a design and reconfigure the FPGA in a non-PR flow, the process reloads all the CRAM bits to a new functionality.

Configuration bitstreams used in a non-PR flow are different than those used in a PR flow. In addition to standard data and CRC check bits, configuration bitstreams for partial reconfiguration also include instructions that direct the PR control block to process the data for partial reconfiguration.

The configuration bitstream written into the CRAM is organized into configuration frames. If a LAB column passes through multiple PR regions, those regions share some programming frames.

1.3.1. SCRUB Mode

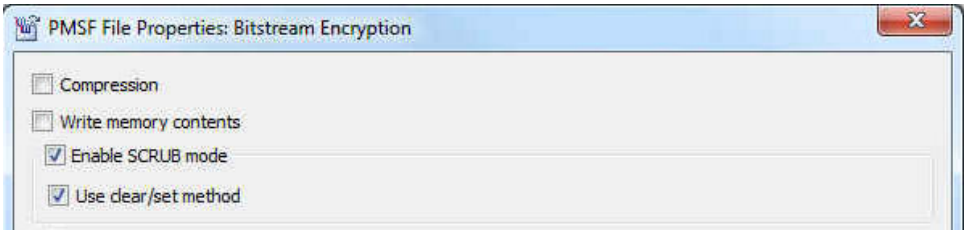
When using SCRUB mode in partial reconfiguration, the unchanging CRAM bits from the static region are "scrubbed" back to their original values.

The static regions controlled by the CRAM bits from the same programming frame as the PR region continue to operate. All the CRAM bits corresponding to a PR region are overwritten with new data, regardless of what was previously contained in the region.

The SCRUB mode of partial reconfiguration involves re-writing all the bits in an entire LAB column of the CRAM, including bits controlling any part of the static region above and below the PR region boundary being reconfigured. You can choose to scrub the values of the CRAM bits to 0, and then rewrite them by turning on the **Use clear/set method** along with **Enable SCRUB mode**. The **Use clear/set method** is the more reliable option, but can increase the size of your bitstream. You can also choose to simply **Enable SCRUB mode**.

Note: You must turn on **Enable SCRUB mode** to use **Use clear/set method**.

Figure 3. Enable SCRUB mode and Use clear/set method

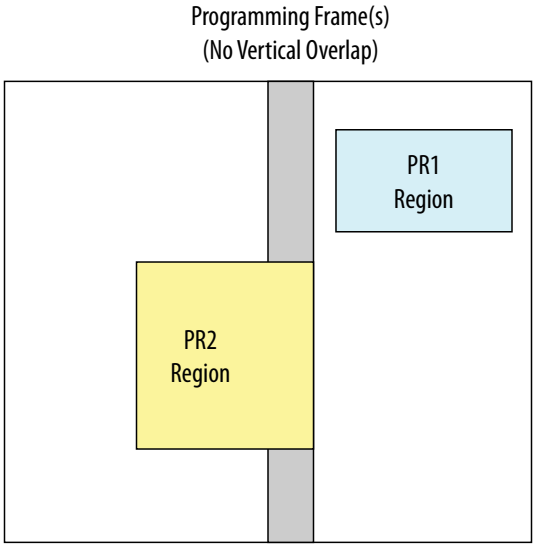


If there are more than one PR regions along a LAB column, and you are trying to reconfigure one of the PR regions, it is not not possible to correctly determine the bits associated with the PR region that is not changing. For this reason, you can not use the SCRUB mode when you have two PR regions that have a vertically overlapping column in the device. This restriction does not apply to the static bits because they never change and you can rewrite them with the same value of the configuration bit.

If you turn on **Enable SCRUB** mode and do not turn on **Use clear/set method**, then the scrub is done in a single pass, writing new values of the CRAM without clearing all the bits first. The advantage of using the SCRUB mode is that the programming file size is much smaller than the AND/OR mode.

Figure 4. SCRUB Mode

This is the floorplan of a FPGA using SCRUB mode, with two PR regions, whose columns do not overlap.



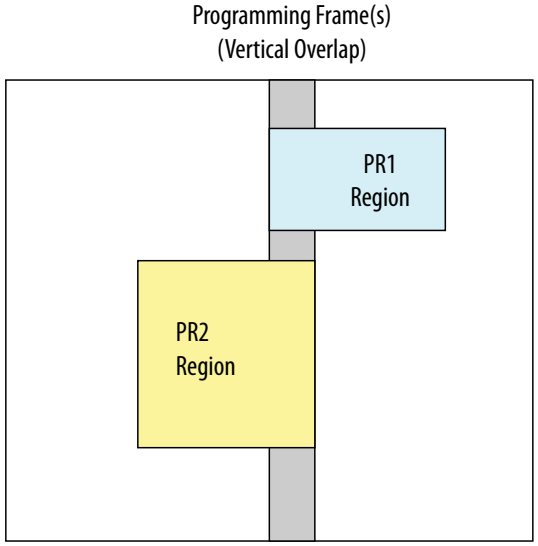
1.3.2. AND/OR Mode

The AND/OR mode refers to how the bits are rewritten. Partial reconfiguration with AND/OR uses a two-pass method.

Simplistically, this can be compared to bits being ANDed with a MASK, and ORed with new values, allowing multiple PR regions to vertically overlap a single column. In the first pass, all the bits in the CRAM frame for a column passing through a PR region are ANDed with '0's while those outside the PR region are ANDed with '1's. After the first pass, all the CRAM bits corresponding to the PR region are reset without modifying the static region. In the second pass for each CRAM frame, new data is ORed with the current value of 0 inside the PR region, and in the static region, the bits are ORed with '0's so they remain unchanged. The programming file size of a PR region using the AND/OR mode could be twice the programming file size of the same PR region using SCRUB mode.

Figure 5. AND/OR Mode

This is the floorplan of a FPGA using AND/OR mode, with two PR regions, with columns that overlap.



Note: If you have overlapping PR regions in your design, you must use AND/OR mode to program all PR regions, including PR regions with no overlap. The Intel Quartus Prime software will not permit the use of SCRUB mode when there are overlapping regions. If none of your regions overlap, you can use AND/OR, SCRUB, or a mixture of both.

1.3.3. Programming File Sizes for a Partial Reconfiguration Project

The programming file size for a partial reconfiguration bitstream is proportional to the area of the PR region.

A partial reconfiguration programming bitstream for AND/OR mode makes two passes on the PR region; the first pass clears all relevant bits, and the second pass sets the necessary bits. Due to this two-pass sequence, the size of a partial bitstream can be larger than a full FPGA programming bitstream depending on the size of the PR region.

When using the AND/OR mode for partial reconfiguration, the formula which describes the approximate file size within ten percent is:

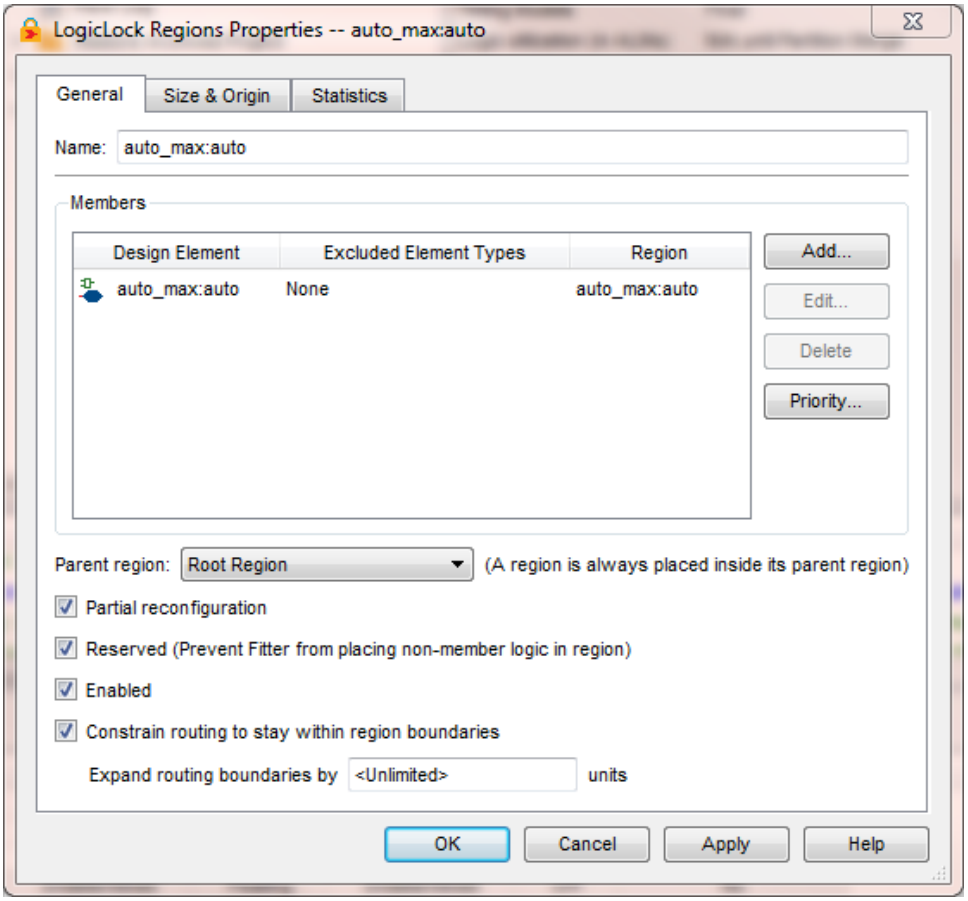
```
PR bitstream size = ((Size of region in the horizontal
direction) /(full horizontal dimension of the part)) * 2 * (size
of full bitstream)
```

The way the Fitter reserves routing for partial reconfiguration increases the effective size for small PR regions from a bitstream perspective. PR bitstream sizes in designs with a single small PR region will not match the file size computed by this equation.

Note: The PR bitstream size is approximately half of the size computed above when using single-pass SCRUB mode. When you use the SCRUB mode with **Use clear/set method** turned on, the bitstream size is comparable to the size calculated for the AND/OR mode.

You can limit expansion of the routing regions in the **LogicLock Regions Properties** dialog box. Alt+L opens the **LogicLock Regions Window**, then right-click on a LogicLock region and click **LogicLock Region Properties**.

Figure 6. LogicLock Regions Properties dialog box



Turn on **Partial reconfiguration**, **Reserved**, **Enabled**, and **Constrain routing to stay within region boundaries**.

You can also control expansion of the routing regions by adding the following two assignments to your Intel Quartus Prime Settings file (`.qsf`):

```
set_global_assignment -name LL_ROUTING_REGION Expanded -
section_id <region name> set_global_assignment -name
LL_ROUTING_REGION_EXPANSION_SIZE 0 -section_id <region name>
```

Adding these to your `.qsf` disables expansion and minimizes the bitstream size.

1.4. Partial Reconfiguration Design Flow

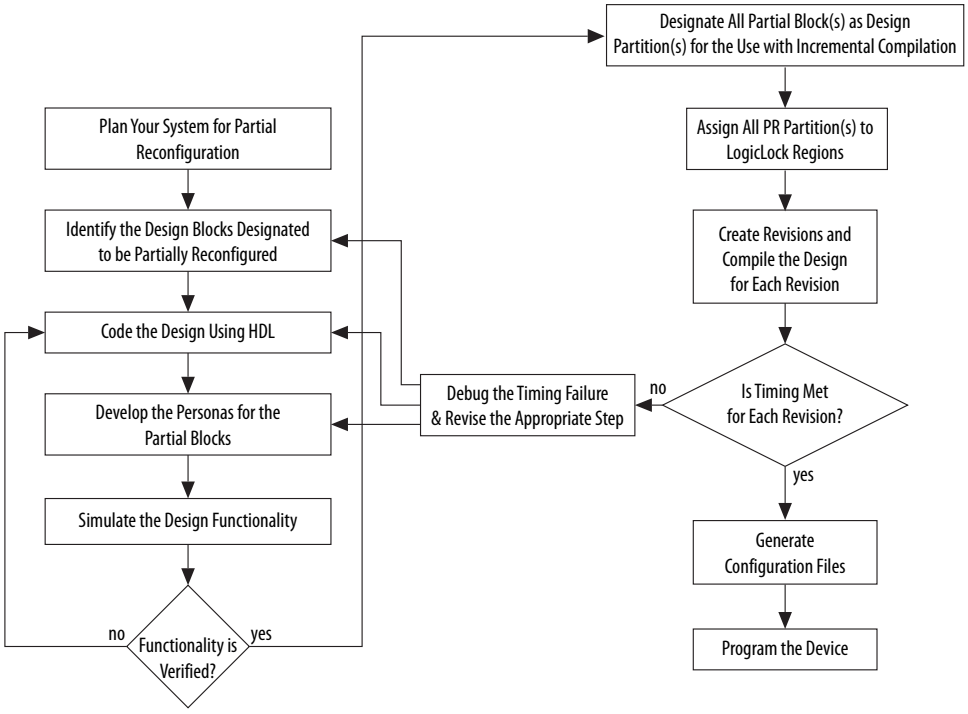
Partial reconfiguration is based on the revision feature in the Intel Quartus Prime software. Your initial design is the base revision, where you define the boundaries of the static region and reconfigurable regions on the FPGA. From the base revision, you create multiple revisions, which contain the static region and describe the differences in the reconfigurable regions.

Two types of revisions are specific to partial reconfiguration: reconfigurable and aggregate. Both import the persona for the static region from the base revision. A reconfigurable revision generates personas for PR regions. An aggregate revision is used to combine personas from multiple reconfigurable revisions to create a complete design suitable for timing analysis.

The design flow for partial reconfiguration also utilizes the Intel Quartus Prime incremental compilation flow. To take advantage of incremental compilation for partial reconfiguration, you must organize your design into logical and physical partitions for synthesis and fitting. The partitions for which partial reconfiguration is enabled (PR partitions) must also have associated LogicLock assignments.

Revisions make use of personas, which are subsidiary archives describing the characteristics of both static and reconfigurable regions, that contain unique logic which implements a specific set of functions to reconfigure a PR region of the FPGA. Partial reconfiguration uses personas to pass this logic from one revision to another.

Figure 7. Partial Reconfiguration Design Flow



The PR design flow requires more initial planning than a standard design flow. Planning requires setting up the design logic for partitioning, and determining placement assignments to create a floorplan. Well-planned partitions can help improve design area utilization and performance, and make timing closure easier. You should also decide whether your system requires partial reconfiguration to originate from the FPGA pins or internally, and which mode you are using; the AND/OR mode or the SCRUB mode, because this influences some of the planning steps described in this section.

You must structure your source code or design hierarchy to ensure that logic is grouped correctly for optimization. Implementing the correct logic grouping early in the design cycle is more efficient than restructuring the code later. The PR flow

requires you to be more rigorous about following good design practices. The guidelines for creating partitions for incremental compilation also include creating partitions for partial reconfiguration.

Use the following best practice guidelines for designing in the PR flow, which are described in detail in this section:

- Determining resources for partial reconfiguration
- Partitioning the design for partial reconfiguration
- Creating incremental compilation partitions for partial reconfiguration
- Instantiating the PR IP core in the design
- Creating wrapper logic for PR regions
- Creating freeze logic for PR regions
- Planning clocks and other global signals for the PR design
- Creating floorplan assignments for the PR design

1.4.1. Design Partitions for Partial Reconfiguration

You must create design partitions for each PR region that you want to partially reconfigure. Optionally, you can also create partitions for the static parts of the design for timing preservation and/or for reducing compilation time.

There is no limit on the number of independent partitions or PR regions you can create in your design. You can designate any partition as a PR partition by enabling that feature in the LogicLock Regions window in the Intel Quartus Prime software.

Partial reconfiguration regions do not support the following IP blocks that require a connection to the JTAG controller:

- In-System Memory Content EditorI
- In-System Signals & Probes
- Virtual JTAG
- Nios II with debug module
- Signal Tap tap or trigger sources

Note: PR partitions can contain only FPGA core resources, they cannot contain I/O or periphery elements.

1.4.2. Incremental Compilation Partitions for Partial Reconfiguration

Use the following best practices guidelines when creating partitions for PR regions in your design:

- Register all partition boundaries; register all inputs and outputs of each partition when possible. This practice prevents any delay penalties on signals that cross partition boundaries and keeps each register-to-register timing path within one partition for optimization.
- Minimize the number of paths that cross partition boundaries.
- Minimize the timing-critical paths passing in or out of PR regions. If there are timing-critical paths that cross PR region boundaries, rework the PR regions to avoid these paths.
- The Intel Quartus Prime software can optimize some types of paths between design partitions for non-PR designs. However, for PR designs, such inter-partition paths are strictly not optimized.

1.4.3. Partial Reconfiguration Controller Instantiation in the Design

Normally you would use the Altera PR IP core to configure the PR process. When you instantiate the PR IP within your PR design, the Stratix V PR control block and the Stratix V CRC block are automatically instantiated in your design. However, you can also write your own custom logic to do the function of the PR IP. In case you are creating your own control logic, or if you are using the PR IP in the external host mode (where in the logic that controls PR process is outside the FPGA undergoing PR operation) , you must instantiate the Stratix V PR control block and the Stratix V CRC block in your design in order to use the PR feature in external host mode. Please refer to the Partial Reconfiguration with an External Host topic for more details.

If you perform PR in internal host mode, you do not have to instantiate the PR control block and the CRC block, since they are are instantiated for you by the PR IP core. Instantiation of the partial reconfiguration controller is required only if your design includes partial reconfiguration in external host mode. Please refer to the *Partial Reconfiguration with an External Host* topic for more details.

When you are manually instantiating the Stratix V Control Block and CRC block, you may want to add the PR control and CRC blocks at the top level of the design.

For example, in a design named Core_Top, all the logic is contained under the Core_Top module hierarchy. Create a wrapper (Chip_Top) at the top-level of the hierarchy that instantiates this Core_Top module, the Stratix V PR control block, and the Stratix V CRC check modules.

If you are performing partial reconfiguration from pins, then the required pins should be on the I/O list for the top-level (Chip_Top) of the project, as shown in the code in the following examples. If you are performing partial reconfiguration from within the core, you may choose another configuration scheme, such as Active Serial, to transmit the reconfiguration data into the core, and then assemble it to 16-bit wide data inside the FPGA within your logic. In such cases, the PR pins are not part of the FPGA I/O.

1.4.3.1. Component Declaration of the PR Control Block and CRC Block in VHDL

To instantiate the PR control block and the CRC block in your design manually, use this code sample containing the component declaration in VHDL. The PR function is performed from within the core (code located in Core_Top) and you must add additional ports to Core_Top to connect to both components. This example is in VHDL but you can create a similar instantiation in Verilog as well.

```
-- The Stratix V control block interface

component stratixv_prblock is
  port(
      clk: in STD_LOGIC := '0';
      corectl: in STD_LOGIC := '0';
      data: in STD_LOGIC_VECTOR(15 downto 0) := (others => '0');
      done: out STD_LOGIC;
      error: out STD_LOGIC;
      externalrequest: out STD_LOGIC;
      prrequest: in STD_LOGIC := '0';
      ready: out STD_LOGIC
      );
end component;

-- The Stratix V CRC block for diagnosing CRC errors

component stratixv_crcblock is
  port(
      shiftnld: in STD_LOGIC ;
      clk: in STD_LOGIC ;
      crcerror: out STD_LOGIC
      );
end component;
```

The following rules apply when connecting the PR control block to the rest of your design:

- The `corectl` signal must be set to '1' (when using partial reconfiguration from core) or to '0' (when using partial reconfiguration from pins).
- The `corectl` signal has to match the **Enable PR pins** option setting in the Device and Pin Options dialog box on the Setting page; if you have turned on **Enable PR pins**, then the `corectl` signal on the PR control block instantiation must be toggled to '0'.
- When performing partial reconfiguration from pins the Intel Quartus Prime software automatically assigns the PR unassigned pins. If you so choose, you can make pin assignments to all the dedicated PR pins in **Pin Planner** or **Assignment Editor**.
- When performing partial reconfiguration from core, you can connect the `prblock` signals to either core logic or I/O pins, excluding the dedicated programming pin such as DCLK.

1.4.3.2. Instantiating the PR Control Block and CRC Block in VHDL

This code example instantiates a PR control block in VHDL, inside your top-level project, Chip_Top:

```
entity Chip_Top is port (
                         --User I/O signals (excluding PR related signals)
                         ..
                         ..
                         );
```

```
end Chip_Top;

-- Following shows the architecture behavior of Chip_Top

m_pr : stratixv_prblock
  port map(
           clk       => dclk,
           corectl   =>'0', --1 - when using PR from inside
                      --0 - for PR from pins; You must also enable
                      -- the appropriate option in Intel Quartus Prime
settings
           prrequest => pr_request,
           data      => pr_data,
           error     => pr_error,
           ready     => pr_ready,
           done      => pr_done
          );
m_crc : stratixv_crcblock
  port map(
          shiftnld      =>  '1',       --If you want to read the EMR register
when
           clk          =>  dummy_clk, --error occurrs, refer to AN539 for the
                                --connectivity forthis 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
          );
```

For more information on port connectivity for reading the Error Message Register (EMR), refer to the following application note.

Related Information

AN539: Test Methodology of Error Detection and Recovery using CRC in Altera FPGA Devices

1.4.3.3. Instantiating the PR Control Block and CRC Block in Verilog HDL

The following example instantiates a PR control block in Verilog HDL, inside your top-level project, Chip_Top:

```
module Chip_Top (
                //User I/O signals (excluding PR related signals)
                ..
                ..
                //PR interface & configuration signals
                  pr_request,
                  pr_ready,
                  pr_done,
                  crc_error,
                  dclk,
                  pr_data,
                  init_done
                 );

//user I/O signal declaration
..
..
//PR interface and configuration signals declaration
  input   pr_request;
  output  pr_ready;
  output  pr_done;
  output  crc_error;
  input  dclk;
  input  [15:0] pr_data;
```

```
  output init_done

stratixv_prblock stratixv_prblock_inst
  (
    .clk      (dclk),
    .corectl  (1'b0),
    .prrequest(pr_request),
    .data     (pr_data),
    .error    (pr_error),
    .ready    (pr_ready),
    .done     (pr_done)
  );

stratixv_crcblock stratixv_crcblock_inst
  (
    .clk      (clk),
    .shiftnld (1'b1),
    .crcerror (crc_error)
  );
endmodule
```

For more information on port connectivity for reading the Error Message Register (EMR), refer to the following application note.

Related Information

AN539: Test Methodology of Error Detection and Recovery using CRC in Altera FPGA Devices

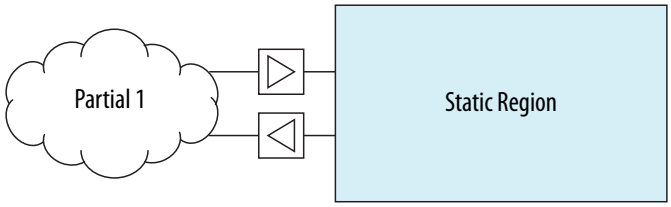
1.4.4. Wrapper Logic for PR Regions

Each persona of a PR region must implement the same input and output boundary ports. These ports act as the boundary between static and reconfigurable logic.

Implementing the same boundary ports ensures that all ports of a PR region remain stationary regardless of the underlying persona, so that the routing from the static logic does not change with different PR persona implementations.

Figure 8. Wire-LUTs at PR Region Boundary

The Intel Quartus Prime software automatically instantiates a wire-LUT for each port of the PR region to lock down the same location for all instances of the PR persona.



If one persona of your PR region has a different number of ports than others, then you must create a wrapper so that the static region always communicates with this wrapper. In this wrapper, you can create dummy ports to ensure that all of the PR personas of a PR region have the same connection to the static region.

The sample code below each create two personas; `persona_1` and `persona_2` are different functions of one PR region. Note that one persona has a few dummy ports. The first example creates partial reconfiguration wrapper logic in Verilog HDL:

```
// Partial Reconfiguration Wrapper in Verilog HDL
module persona    //this module is persona_1
  (
```

```
   input reset,
   input [2:0] a,
   input [2:0] b,
   input [2:0] c,
   output [3:0] p,
   output [7:0] q
  );
reg [3:0] p, q;
always@(a or b)
  begin
    p = a + b ;
  end

always@(a or b or c or p)
  begin
    q = (p*a - b*c )
  end
endmodule

module persona    //this module is persona_2
(
 input reset,
 input [2:0] a,
 input [2:0] b,
 input [2:0] c,   //never used in this persona
 output [3:0] p,
 output [7:0] q   //never assigned in this persona
);
reg [3:0] p, q;
always@(a or b)
  begin
    p = a * b;    // note q is not assigned value in this persona
  end
endmodule
```

The following example creates partial reconfiguration wrapper logic in VHDL.

```
-- Partial Reconfiguration Wrapper in VHDL
-- this module is persona_1
entity persona is
  port(
       a:in STD_LOGIC_VECTOR (2 downto 0);
      b:in STD_LOGIC_VECTOR (2 downto 0);
      c:in STD_LOGIC_VECTOR (2 downto 0);
      p: out STD_LOGIC_VECTOR (3 downto 0);
      q: out STD_LOGIC_VECTOR (7 downto 0)
       );
end persona;

architecture synth of persona is
    begin
      process(a,b)
        begin
        p <= a + b;
       end process;

     process (a, b, c, p)
        begin
          q <= (p*a - b*c);
        end process;
end synth;

-- this module is persona_2
entity persona is
  port(
       a:in STD_LOGIC_VECTOR (2 downto 0);
       b:in STD_LOGIC_VECTOR (2 downto 0);
       c:in STD_LOGIC_VECTOR (2 downto 0);   --never used in this persona
       p:out STD_LOGIC_VECTOR (3 downto 0);
       q:out STD_LOGIC_VECTOR (7 downto 0) --never used in this persona
```

```
      );
end persona_2;

architecture synth of persona_2 is
  begin
    process(a, b)
      begin
        p <= a *b; --note q is not assigned a value in this persona
     end process;
end synth;
```

1.5. Freeze Logic for PR Regions

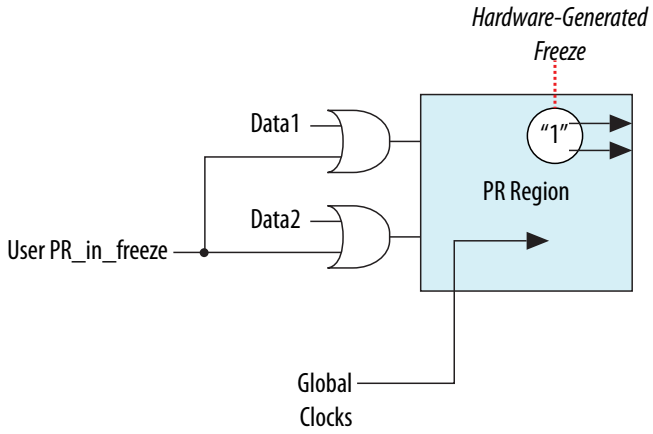
When you use partial reconfiguration, you must freeze all non-global inputs of a PR region except global clocks. Locally routed signals are not considered global signals, and must also be frozen during partial reconfiguration. Freezing refers to driving a '1' on those PR region inputs. When you start a partial reconfiguration process, the chip is in user mode, with the device still running.

When you instantiate the Altera PR IP core in your design, the IP incudes a freeze port which you can use to freeze the non-global inputs of the PR region. In case your design has multiple PR regions, you must create decoding logic to freeze only the inputs of the PR region being partially reconfigured.

If you are not using the Altera PR IP, you must include logic to freeze the inputs of the PR regions in the design as required for proper operation.

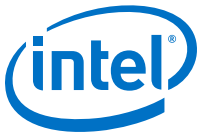
Freezing all non-global inputs for the PR region ensures there is no contention between current values that may result in unexpected behavior of the design after partial reconfiguration is complete. Global signals going into the PR region should not be frozen to high. The Intel Quartus Prime software freezes the outputs from the PR region; therefore the logic outside of the PR region is not affected.

Figure 9. Freezing at PR Region Boundary



During partial reconfiguration, the static region logic should not depend on the outputs from PR regions to be at a specific logic level for the continued operation of the static region.

The easiest way to control the inputs to PR regions is by creating a wrapper around the PR region in RTL. In addition to freezing all inputs high, you can also drive the outputs from the PR block to a specific value, if required by your design. For example,



if the output drives a signal that is active high, then your wrapper could freeze the output to GND. The idea is to make sure the static region will not stall or go to indeterminate state, when the PR region is getting a new persona through PR.

The following example implements a freeze wrapper in Verilog HDL, on a module named `pr_module`.

```
module freeze_wrapper
(
 input reset,      // global reset signal
 input freeze,     // PR process active, generated by user logic
 input clk1,       // global clock signal
 input clk2,       // non-global clock signal
 input [3:0] control_mode,
 input [3:0] framer_ctl,
 output [15:0] data_out
);
wire [3:0]control_mode_wr, framer_ctl_wr;
wire clk2_to_wr;
//instantiate pr_module
pr_module pr_module
(
 .reset (reset),                  //input
 .clk1 (clk1),                    //input, global clock
 .clk2 (clk2_to_wr),              // input, non-global clock
 .control_mode (control_mode_wr), //input
 .framer_ctl (framer_ctl_wr),     //input
 .pr_module_out (data_out)        // collection of outputs from pr_module
);

// Freeze all inputs

assign control_mode_wr = freeze ? 4'hF: control_mode;
assign framer_ctl_wr = freeze ? 4'hF: framer_ctl;
assign clk2_to_wr = freeze ? 1'b1 : clk2;

endmodule
```

The following example implements a freeze wrapper in VHDL, on a module named `pr_module`.

```
entity freeze_wrapper is
port(
     reset:in STD_LOGIC;      -- global reset signal
     freeze:in STD_LOGIC;
     clk1: in STD_LOGIC;      -- global signal
     clk2: in STD_LOGIC;      -- non-global signal
     control_mode: in STD_LOGIC_VECTOR (3 downto 0);
     framer_ctl: in STD_LOGIC_VECTOR (3 downto 0);
     data_out: out STD_LOGIC_VECTOR (15 downto 0)
    );
end freeze_wrapper;

architecture behv of freeze_wrapper is

  component pr_module
  port(
       reset:in STD_LOGIC;
       clk1:in STD_LOGIC;
       clk2:in STD_LOGIC;
       control_mode:in STD_LOGIC_VECTOR (3 downto 0);
       framer_ctl:in STD_LOGIC_VECTOR (3 downto 0);
       pr_module_out:out STD_LOGIC_VECTOR (15 downto 0)
      );
  end component

  signal control_mode_wr: in STD_LOGIC_VECTOR (3 downto 0);
  signal framer_ctl_wr : in STD_LOGIC_VECTOR (3 downto 0);
```

```
  signal clk2_to_wr : STD_LOGIC;
  signal data_out_temp : STD_LOGIC_VECTOR (15 downto 0);
  signal logic_high : STD_LOGIC_VECTOR (3 downto 0):="1111";

  begin
    data_out(15 downto 0) <= data_out_temp(15 downto 0);

  m_pr_module: pr_module
     port map (
               reset => reset,
               clk1 => clk1,
               clk2 => clk2_to_wr,
               control_mode =>control_mode_wr,
               framer_ctl => framer_ctl_wr,
               pr_module_out => data_out_temp);
               -- freeze all inputs

  control_mode_wr <= logic_high when (freeze ='1') else control_mode;
  framer_ctl_wr <= logic_high when (freeze ='1') else framer_ctl;
  clk2_to_wr <= logic_high(0) when (freeze ='1') else clk2;

end architecture;
```

1.5.1. Clocks and Other Global Signals for a PR Design

For non-PR designs, the Intel Quartus Prime software automatically promotes high fan-out signals onto dedicated clocks or other forms of global signals during the pre-fitter stage of design compilation using a process called global promotion. For PR designs, however, automatic global promotion is disabled by default for PR regions, and you must assign the global clock resources necessary for PR partitions. Clock resources can be assigned by making Global Signal assignments in the Intel Quartus Prime Assignment Editor, or by adding Clock Control Block (altclkctrl) IP core blocks in the design that drive the desired global signals.

There are 16 global clock networks in a Stratix V device. However, only six unique clocks can drive a row clock region limiting you to a maximum of six global signals in each PR region. The Intel Quartus Prime software must ensure that any global clock can feed every location in the PR region.

The limit of six global signals to a PR region includes the GCLK, QCLK and PCLKs used inside of the PR region. Make QSF assignments for global signals in your project's Intel Quartus Prime Settings File (`.qsf`), based on the clocking requirements for your design. In designs with multiple clocks that are external to the PR region, it may be beneficial to align the PR region boundaries to be within the global clock boundary (such as QCLK or PCLK).

If your PR region requires more than six global signals, modify the region architecture to reduce the number of global signals within this to six or fewer. For example, you can split a PR region into multiple regions, each of which uses only a subset of the clock domains, so that each region does not use more than six.

Every instance of a PR region that uses the global signals (for example, PCLK, QCLK, GCLK, ACLR) must use a global signal for that input.

Global signals can only be used to route certain secondary signals into a PR region and the restrictions for each block are listed in the following table. Data signals and other secondary signals not listed in the table, such as synchronous clears and clock enables are not supported.

Table 2. Supported Signal Types for Driving Clock Networks in a PR Region

Block Types	Supported Signals for Global/Periphery/Quadrant Clock Networks
LAB	`Clock, ACLR`
RAM	`Clock, ACLR, Write Enable(WE), Read Enable(RE)`
DSP	`Clock, ACLR`

Note: PR regions are allowed to contain output ports that are used outside of the PR region as global signals.

- If a global signal feeds both static and reconfigurable logic, the restrictions in the table also apply to destinations in the static region. For example, the same global signal cannot be used as an SCLR in the static region and an ACLR in the PR region.
- A global signal used for a PR region should only feed core blocks inside and outside the PR region. In particular you should not use a clock source for a PR region and additionally connect the signal to an I/O register on the top or bottom of the device. Doing so may cause the Assembler to give an error because it is unable to create valid programming mask files.

1.5.2. Floorplan Assignments for PR Designs

You must create a LogicLock region so the interface of the PR region with the static region is the same for any persona you implement. If different personas of a PR region have different area requirements, you must make a LogicLock region assignment that contains enough resources to fit the largest persona for the region. The static regions in your project do not necessarily require a floorplan, but depending on any other design requirement, you may choose to create a floorplan for a specific static region. If you create multiple PR regions, and are using SCRUB mode, make sure you have one column or row of static region between each PR region.

There is no minimum or maximum size for the LogicLock region assigned for a PR region. Because wire-LUTs are added on the periphery of a PR region by the Intel Quartus Prime software, the LogicLock region for a PR region must be slightly larger than an equivalent non-PR region. Make sure the PR regions include only the resources that can be partially reconfigured; LogicLock regions for PR can only contain only LABs, DSPs, and RAM blocks. When creating multiple PR regions, make sure there is at least one static region column between each PR region. When multiple PR regions are present in a design, the shape and alignment of the region determines whether you use the SCRUB or AND/OR PR mode.

You can use the default **Auto size** and **Floating location** LogicLock region properties to estimate the preliminary size and location for the PR region.

You can also define regions in the floorplan that match the general location and size of the logic in each partition. You may choose to create a LogicLock region assignment that is non-rectangular, depending on the design requirements, but disjoint LogicLock regions are not allowed for PR regions in your first compilation of the project.

After compilation, use the Fitter-determined size and origin location as a starting point for your design floorplan. Check the quality of results obtained for your floorplan location assignments and make changes to the regions as needed.

Alternatively, you can perform Analysis and Synthesis, and then set the regions to the required size based on resource estimates. In this case, use your knowledge of the connections between partitions to place the regions in the floorplan.

1.6. Implementation Details for Partial Reconfiguration

This section describes implementation details that help you create your PR design.

1.6.1. Interface with the PR Control Block through a PR Host

During partial reconfiguration, a PR bitstream stored outside the FPGA being partially reconfigured must be sent to the PR Control Block in the FPGA. This enables the control block to update the CRAM bits necessary to configure the PR region in the FPGA.

Two scenarios are possible, depending on whether the control logic to transfer the bitstream is located within the FPGA or outside the FPGA being reconfigured.

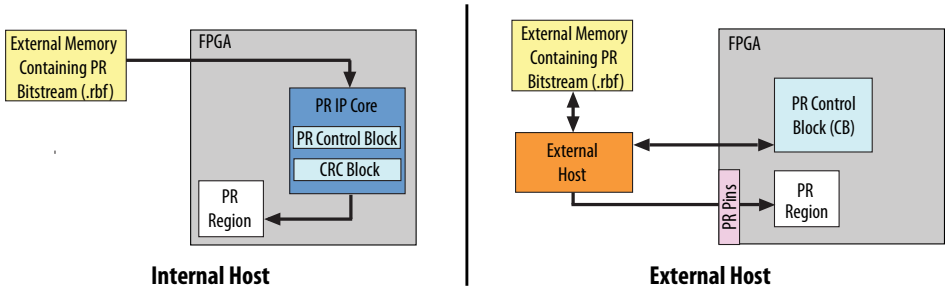
- If the PR IP core is instantiated inside the FPGA being reconfigured, it is termed PR with an internal host; the Altera PR IP core helps you perform the transfer of the PR bitstream.
- When the PR IP is instantiated outside the FPGA being reconfigured, it is termed as PR with an external host.

There is a well-defined interface and a specific protocol to transfer the PR bitstream from the external bitstream source to the PR control block. When you use the Altera PR IP core, the protocol requirements are automatically met by the IP.

It is also possible to write your own control logic, or use a Nios® processor to do this PR bitstream transfer. Note that when create your own control logic for the PR Host, you must make sure to meet the interface requirements described later in this chapter.

Figure 10. Managing Partial Reconfiguration with an Internal or External Host

The figure shows how these blocks should be connected to the PR control block (CB). In your system, you will have either the External Host or the Internal Host, but not both. The external host can be implemented by instantiating the PR IP core outside the FPGA being reconfigured, may be in another Altera FPGA, or processor/PC (PR over PCIe) , or can be implemented by user logic.



The PR mode is independent of the full chip programming mode. You can use any of the supported full chip configuration modes for configuring the full FPGA for your PR design.

If you are creating your own custom logic for implementing a PR internal host, you can use any interface to load the PR bitstream data to the FPGA; for example, from a serial or a parallel flash device; and then format the PR bitstream data to match the FPPx16 interface on the PR Control Block.

When using an external host, you must implement the control logic for managing system aspects of partial reconfiguration on an external device. To use the external host for your design, turn on the **Enable PR Pins** option in the **Device and Pin Options** dialog box in the Intel Quartus Prime software when you compile your design. If this setting is turned off, then you must use an internal host. Also, you must tie the `corectl` port on the PR control block instance in the top-level of the design to the appropriate level for the selected mode.

Related Information

Partial Reconfiguration Pins on page 23
Partial Reconfiguration Dedicated Pins Table

1.6.2. Partial Reconfiguration Pins

Partial reconfiguration can be performed through external pins or from inside the core of the FPGA.

When using PR from pins, some of the I/O pins are dedicated for implementing partial reconfiguration functionality. If you perform partial reconfiguration from pins, then you must use the passive parallel with 16 data bits (FPPx16) configuration mode. All dual-purpose pins should also be specified to **Use as regular I/O**.

To enable partial reconfiguration from pins in the Intel Quartus Prime software, perform the following steps:

1. From the Assignments menu, click **Device**, then click **Device and Pin Options**.
2. In the **Device and Pin Options** dialog box, select **Partial Reconfiguration** in the **Category** list and turn on **Enable PR pins** from the **Options** list.
3. Click **Configuration** in the **Category** list and select **Passive Parallel x16** from the **Configuration scheme** list.
4. Click **Dual-Purpose Pins** in the **Category** list and verify that all pins are set to **Use as regular I/O** rather than **Use as input tri-stated**.
5. Click **OK**, or continue to modify other settings in the **Device and Pin Options** dialog box.
6. Click **OK**.

Note: You can enable open drain on PR pins from the **Device and Pin Options** dialog box in the **Partial Reconfiguration** dialog box.

Table 3. Partial Reconfiguration Dedicated Pins Description

Pin Name	Pin Type	Pin Description
PR_REQUEST	Input	Dedicated input when **Enable PR pins** is turned on; otherwise, available as user I/O. Logic high on pin indicates the PR host is requesting partial reconfiguration.
PR_READY	Output	Dedicated output when **Enable PR pins** is turned on; otherwise, available as user I/O. Logic high on this pin indicates the Stratix V control block is ready to begin partial reconfiguration.

continued...

Pin Name	Pin Type	Pin Description
PR_DONE	Output	Dedicated output when **Enable PR pins** is turned on; otherwise, available as user I/O. Logic high on this pin indicates that partial reconfiguration is complete.
PR_ERROR	Output	Dedicated output when **Enable PR pins** is turned on; otherwise, available as user I/O. Logic high on this pin indicates the device has encountered an error during partial reconfiguration.
DATA[15:0]	Input	Dedicated input when **Enable PR pins** is turned on; otherwise available as user I/O. These pins provide connectivity for PR_DATA to transfer the PR bitstream to the PR Controller.
DCLK	Bidirectional	Dedicated input when **Enable PR pins** is turned on; PR_DATA is sent synchronous to this clock.

For more information on different configuration modes for Stratix V devices, and specifically about FPPx16 mode, refer to the *Configuration, Design Security, and Remote System Upgrades in Stratix V Devices* chapter of the *Stratix V Handbook*.

Related Information

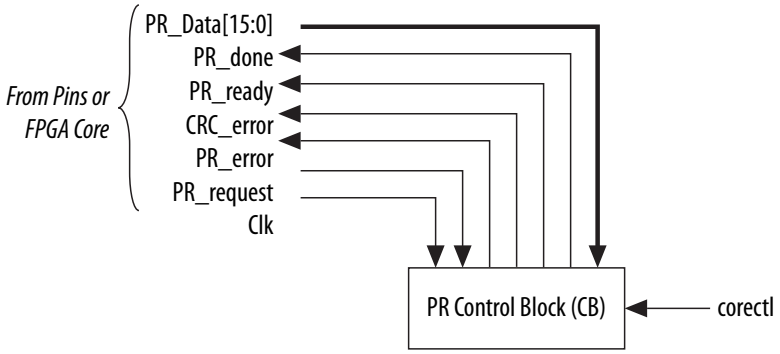
Configuration, Design Security, and Remote System Upgrades in Stratix V Devices

1.6.3. PR Control Signals Interface

You can use the Intel Quartus Prime **Assembler** and the **Convert Programming File** utilities to generate the different bitstreams necessary for full chip configuration and for partial reconfiguration. The programming bit-stream for partial reconfiguration contains the instructions (opcodes) as well as the configuration bits, necessary for reconfiguring each of the partial regions. When using an external host, the interface ports on the control block are mapped to FPGA pins. When using an internal host, these signals are within the core of the FPGA. When using the PR IP core as an internal host, connect the signals on the PR IP core appropirately as described in the Partial Reconfiguration IP Core User Guide and follow the instructions to start the PR process on the FPGA. If you are not using the PR IP core, make sure you understand these PR interface signals.

Figure 11. Partial Reconfiguration Interface Signals

These handshaking control signals are used for partial reconfiguration.



- PR_DATA: The configuration bitstream is sent on PR_ DATA[15:0], synchronous to the Clk.
- PR_DONE: Sent from CB to control logic indicating the PR process is complete.
- PR_READY: Sent from CB to control logic indicating the CB is ready to accept PR data from the control logic.
- CRC_Error: The CRC_Error generated from the device's CRC block, is used to determine whether to partially reconfigure a region again, when encountering a CRC_Error.
- PR_ERROR: Sent from CB to control logic indicating an error during partial reconfiguration.
- PR_REQUEST: Sent from your control logic to CB indicating readiness to begin the PR process.
- corectl: Determines whether partial reconfiguration is performed internally or through pins.

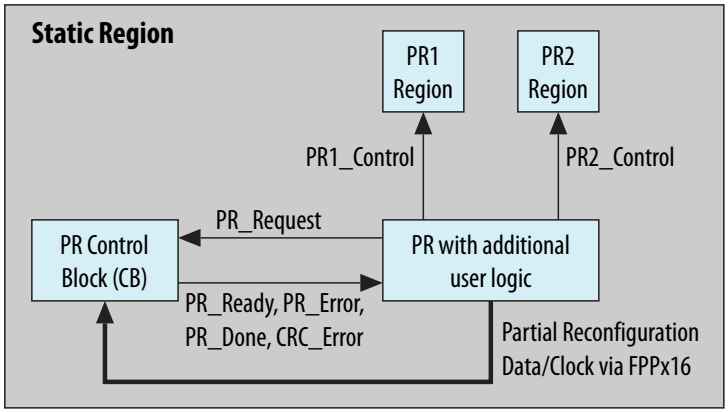
1.6.4. Reconfiguring a PR Region

The figure below shows an internal host for PR, where the PR IP core is implemented inside the FPGA. However, these principles are also applicable for partial reconfiguration with an external host.

The PR control block (CB) represents the Stratix V PR controller inside the FPGA. PR1 and PR2 are two PR regions in a user design. In addition to the four control signals (PR_REQUEST, PR_READY, PR_DONE, PR _ERROR) and the data/clock signals interfacing with the PR control block, your PR Control IP should also send a control signal (PR_CONTROL) to each PR region. This signal implements the freezing and unfreezing of the PR Interface signals. This is necessary to avoid contention on the FPGA routing fabric. In a case such as this, you need to add some decoding logic in the design, in addition to instantiating the PR IP core.

Figure 12. Example of a PR System with Two PR Regions

Implementation of PR Control logic in the FPGA.



After the FPGA device has been configured with a full chip configuration at least once, the INIT_DONE signal is released, and the signal is asserted high due to the external resistor on this pin. The INIT_DONE signal must be assigned to a pin to monitor it externally. When a full chip configuration is complete, and the device is in user mode, the following steps describe the PR sequence:

1. Begin a partial reconfiguration process from your PR Control logic, which initiates the PR process for one or more of the PR regions (asserting PR1_Control or PR2_Control in the figure). The wrapper HDL described earlier freezes (pulls high) all non-global inputs of the PR region before the PR process.
2. If you are using the PR IP core, use the PR_START signal to start reconfiguring the PR region. When you are not using the PR IP core, your control logic should send the PR_REQUEST signal from your control logic to the PR Control Block (CB). If your design uses an external controller, monitor INIT_DONE to verify that the chip is in user mode before asserting the PR_START or PR_REQUEST signal. The CB initializes itself to accept the PR data and clock stream. After that, the CB asserts a PR_READY signal to indicate it can accept PR data. If you are using the PR IP, the timing relationsips between the control and data signals is managed by the IP core. Data and clock signals are sent to the PR control block to partially reconfigure the PR region interface.

   Note: If you write your own controller logic, specify that exactly four clock-cycles must occur before sending the PR data to make sure the PR process progresses correctly.

   - When there are multiple PR personas for the PR region, your control logic must determine the programming file data for partial reconfiguration and specify the correct file.
   - When there are multiple PR regions in the design, then your control logic determines which regions require reconfiguration based on system requirements.
   - At the end of the PR process, the PR control block asserts a PR_DONE signal and deasserts the PR_READY signal. The Altera PR IP core further processes these signals to assert a 3-bit status signal. If you are not using the Altera PR IP, your design must take approcpriate action as defined by the timing diagrams when PR_DONE is asserted.
   - If you want to suspend sending data, you can implement logic to pause the clock at any point.
3. When you are not using the PR IP core, your custom control logic must deassert the PR_REQUEST signal within eight clock cycles after the PR_DONE signal goes high. If your logic does not deassert the PR_REQUEST signal within eight clock cycles, a new PR cycle starts.
4. If your design includes additional PR regions, repeat steps 2 – 3 for each region. Otherwise, proceed to step 5.
5. When you are not using the PR IP core, your custom control logic must deassert the PR_CONTROL signal(s) to the PR region. The freeze wrapper releases all input signals of the PR region, thus the PR region is ready for normal user operation.
6. You must perform a reset cycle to the PR region to bring all logic in the region to a known state. After partial reconfiguration is complete for a PR region, the states in which the logic in the region come up is unknown.

The PR event is now complete, and you can resume operation of the FPGA with the newly configured PR region.

At any time after the start of a partial reconfiguration cycle, the PR host can suspend sending the PR_DATA, but the host must suspend sending the PR_CLK at the same time. If the PR_CLK is suspended after a PR process, there must be at least 20 clock cycles after the PR_DONE or PR_ERROR signal is asserted to prevent incorrect behavior.

For an overview of different reset schemes in Altera devices, refer to the *Recommended Design Practices* chapter in the *Intel Quartus Prime Handbook*.

Related Information

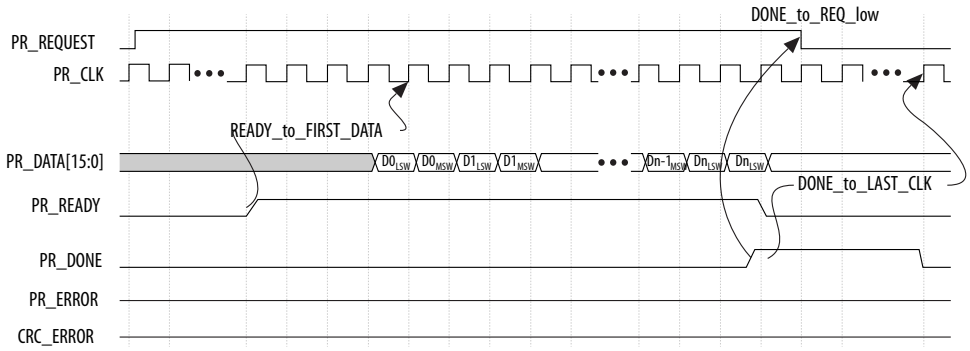
Partial Reconfiguration Cycle Waveform on page 27
For more information on clock requirements for partial reconfiguration.

1.6.5. Partial Reconfiguration Cycle Waveform

When you are using the Altera PR IP in the internal host mode, all the timing relations between various interface signals are met by default, and you can skip reading this section. If you are using PR with an external host or implementing your own custom PR internal host logic, pay attention to these timing relationships when designing your logic. The PR host initiates the PR request, transfers the data to the FPGA device when it is ready, and monitors the PR process for any errors or until it is done.

A PR cycle is initiated by the host (internal or external) by asserting the PR_REQUEST signal high. When the FPGA device is ready to begin partial reconfiguration, it responds by asserting the PR_READY signal high. The PR host responds by sending configuration data on DATA [15:0]. The data is sent synchronous to PR_CLK. When the FPGA device receives all PR data successfully, it asserts the PR_DONE high, and de-asserts PR_READY to indicate the completion of the PR cycle. The PR host must monitor the PR process until either the successful completion of PR (indicated by PR_DONE), or an error condition is asserted.

Figure 13. Partial Reconfiguration Timing Diagram



If there is an error encountered during partial reconfiguration, the FPGA device asserts the PR_ERROR signal high and de-asserts the PR_READY signal low.

Whenever either of these two signals are asserted, the host must de-assert PR_REQUEST within eight PR_CLK cycles. As a response to PR_ERROR error, the host can optionally request another partial reconfiguration or perform a full FPGA configuration.

To prevent incorrect behavior, the PR_CLK signal must be active a minimum of twenty clock cycles after PR_DONE or PR_ERROR signal is asserted high. Once PR_DONE is asserted, PR_REQUEST must be de-asserted within eight clock cycles. PR_DONE is de-asserted by the device within twenty PR_CLK cycles. The host can assert PR_REQUEST again after the 20 clocks after PR_DONE is de-asserted.

Table 4. Partial Reconfiguration Clock Requirements

Signal timing requirements for partial reconfiguration.

Timing Parameters	Value (clock cycles)
PR_READY to first data	4 (exact)
PR_ERROR to last clock	20 (minimum)
PR_DONE to last clock	20 (minimum)
DONE_to_REQ_low	8 (maximum)
Compressed PR_READY to first data	4 (exact)
Encrypted PR_READY to first data (when using double PR)	8 (exact)
Encrypted and Compressed PR_READY to first data (when using double PR)	12 (exact)

At any time during partial reconfiguration, to pause sending PR_DATA, the PR host can stop toggling PR_CLK. The clock can be stopped either high or low.

At any time during partial reconfiguration, the PR host can terminate the process by de-asserting the PR request. A partially completed PR process results in a PR error. You can have the PR host restart the PR process after a failed process by sending out a new PR request 20 cycles later.

If you terminate a PR process before completion, and follow it up with a full FPGA configuration by asserting nConfig, then you must toggle PR_CLK for an additional 20 clock cycles prior to asserting nConfig to flush the PR_CONTROL_BLOCK and avoid lock up.

During these steps, the PR control block might assert a PR_ERROR or a CRC_ERROR signal to indicate that there was an error during the partial reconfiguration process. Assertion of PR_ERROR indicates that the PR bitstream data was corrupt, and the assertion of CRC error indicates a CRAM CRC error either during or after completion of PR process. If the PR_ERROR or CRC_ERROR signals are asserted, you must plan whether to reconfigure the PR region or reconfigure the whole FPGA, or leave it unconfigured.

Important: The PR_CLK signal has different a nominal maximum frequency for each device. Most Stratix V devices have a nominal maximum frequency of at least 62.5 MHz.

1.7. Example of a Partial Reconfiguration Design with an External Host

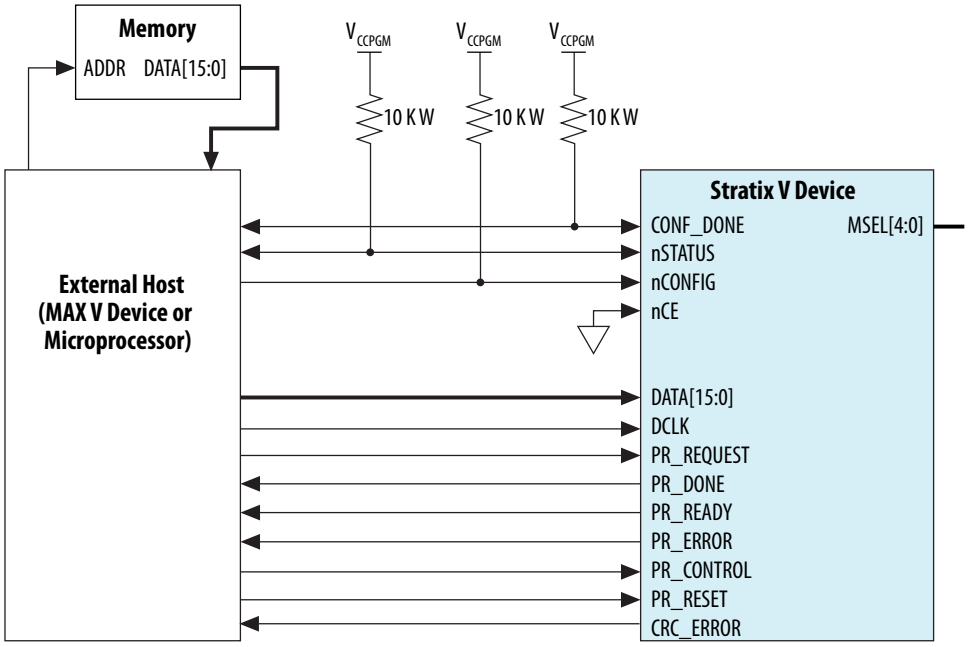
For partial reconfiguration using an external host, you must set the MSEL [4:0] pins for FPPx16 configuration scheme.

You can use Altera PR IP implemented by another supported Altera FPGA device, implement your own control logic in an FPGA or CPLD, or use a microcontroller to implement the configuration and PR controller. In this setup, shown in the following figure, the Stratix V device configures in FPPx16 mode during power-up. Alternatively, you can use a JTAG interface to configure the Stratix V device.

At any time during user-mode, the external host can initiate partial reconfiguration and monitor the status using the external PR dedicated pins: PR_REQUEST, PR_READY, PR_DONE, and PR_ERROR. In this mode, the external host must respond appropriately to the hand-shaking signals for a successful partial reconfiguration. This includes acquiring the data from the flash memory and loading it into the Stratix V device on DATA[15:0].

Figure 14. Connecting to an External Host

The connection setup for partial reconfiguration with an external host in the FPPx16 configuration scheme.



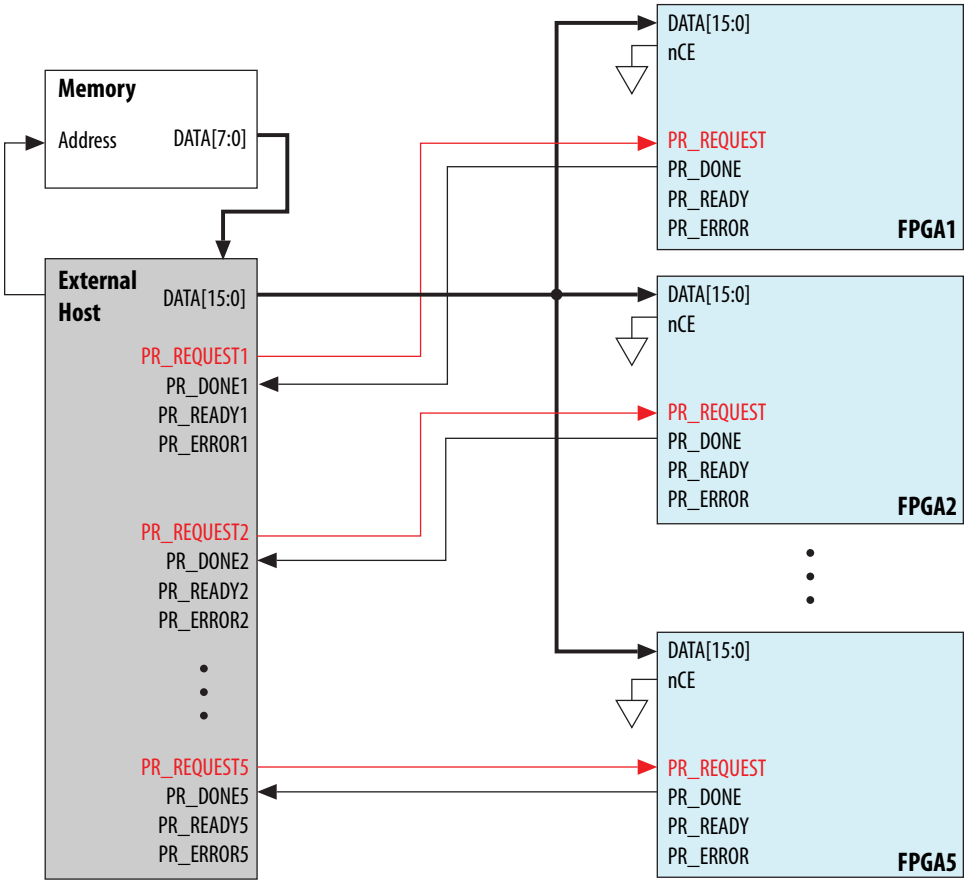
Note: If you don't care to write an external host controller, you can implement an external host with the Partial Reconfiguration IP core on a MAX 10 or other FPGA device.

1.7.1. Example of Using an External Host with Multiple Devices

You must design the external host to accommodate the arbitration scheme that is required for your system, as well as the partial reconfiguration interface requirement for each device.

Figure 15. Connecting Multiple FPGAs to an External Host

An example of an external host controlling multiple Stratix V devices on a board.



1.8. Example Partial Reconfiguration with an Internal Host

You can create PR internal host logic with the PR IP core. If your design uses an internal host, the PR IP core handles the required hand-shaking protocol with the PR control block.

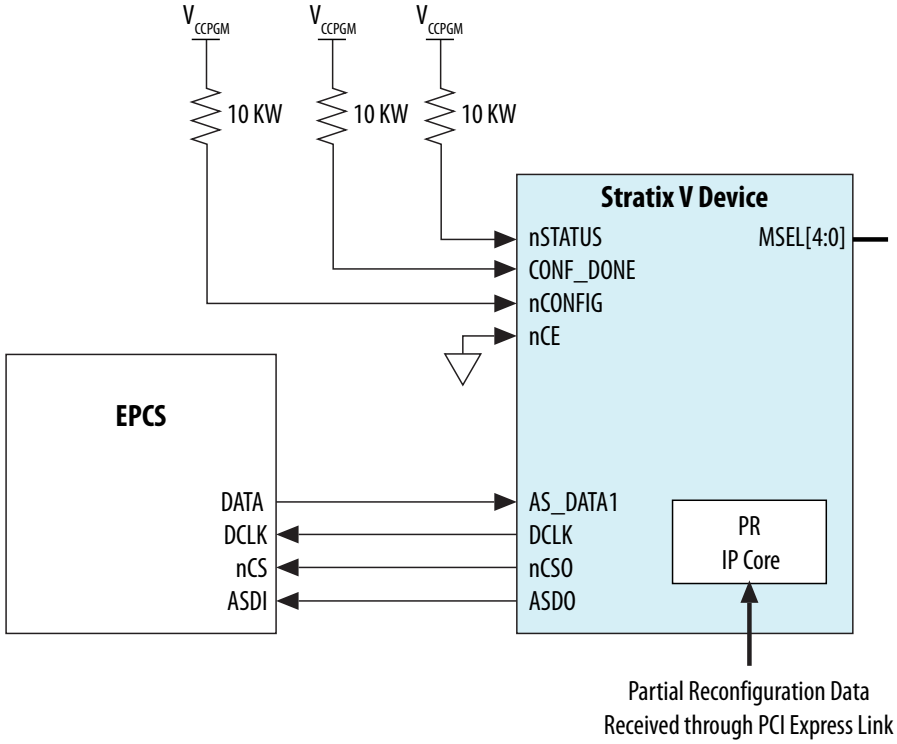
The PR programming bitstream(s) stored in an external flash device can be routed through the regular I/Os of the FPGA device, or received through the high speed transceiver channel (PCI Express, SRIO or Gigabit Ethernet), for processing by the internal host.

The PR dedicated pins (PR_REQUEST, PR_READY, PR_DONE, and PR_ERROR) can be used as regular I/Os when performing partial reconfiguration with an internal host. For the full FPGA configuration upon power-up, you can set the `MSEL[4:0]` pins to match the configuration scheme, for example, Active Serial, Passive Serial, FPPx8, FPPx16, or FPPx32. Alternatively, you can use the JTAG interface to configure the FPGA device. At any time during user-mode, you can initiate partial reconfiguration through the FPGA core fabric using the PR internal host.

In the following figure, the programming bitstream for partial reconfiguration is received through the PCI Express link, and your logic converts the data to the FPPx16 mode.

Figure 16. Connecting to an Internal Host

An example of the configuration setup when performing partial reconfiguration using the internal host.



1.9. Partial Reconfiguration Project Management

When compiling your PR project, you must create a base revision, and one or more reconfigurable revisions. The project revision you start out is termed the base revision.

1.9.1. Create Reconfigurable Revisions

To create a reconfigurable revision, use the **Revisions** tab of the **Project Navigator** window in the Intel Quartus Prime software. When you create a reconfigurable revision, the Intel Quartus Prime software adds the required assignments to associate the reconfigurable revision with the base revision of the PR project. You can add the necessary files to each revision with the **Add/Remove Files** option in the **Project** option under the **Project** menu in the Intel Quartus Prime software. With this step, you can associate the right implementation files for each revision of the PR project.

Important: You must use the **Revisions** tab of the **Project Navigator** window in the Intel Quartus Prime software when creating revisions for partial reconfiguration. Revisions created using **Project ➤ Revisions** cannot be reconfigured.

1.9.2. Compiling Reconfigurable Revisions

Altera recommends that you use the largest persona of the PR region for the base compilation so that the Intel Quartus Prime software can automatically budget sufficient routing.

Here are the typical steps involved in a PR design flow.

1. Compile the base revision with the largest persona for each PR region.
2. Create reconfigurable revisions for other personas of the PR regions by right-clicking in the **Revisions** tab in the Project Navigator.
3. Compile your reconfigurable revisions.
4. Analyze timing on each reconfigurable revision to make sure the design performs correctly to specifications.
5. Create aggregate revisions as needed.
6. Create programming files.

For more information on compiling a partial reconfiguration project, refer to *Performing Partial Reconfiguration* in Intel Quartus Prime Help.

1.9.3. Timing Closure for a Partial Reconfiguration Project

As with any other FPGA design project, simulate the functionality of various PR personas to make sure they perform to your system specifications. You must also make sure there are no timing violations in the implementation of any of the personas for every PR region in your design project.

In the Intel Quartus Prime software, this process is manual, and you must run multiple timing analyses, on the base, reconfigurable, and aggregate revisions. The different timing requirements for each PR persona can be met by using different SDC constraints for each of the personas.

The interface between the partial and static partitions remains identical for each reconfigurable and aggregate revision in the PR flow. If all the interface signals between the static and the PR regions are registered, and there are no timing violations within the static region as well as within the PR regions, the reconfigurable and aggregate revisions should not have any timing violations.

However, you should perform timing analysis on the reconfigurable and aggregate revisions, in case you have any unregistered signals on the interface between partial reconfiguration and static regions.

1.9.4. PR Bitstream Compression and Encryption (Intel Arria® 10 Designs)

You can compress and encrypt the base bitstream and the PR bitstream for your 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 PR images only when the base image is encrypted.
- The Encryption Key Programming (`.ekp`) file generates when encrypting the base image and must be used for encrypting the PR bitstream.
- When you compress the bitstream, present each `PR_DATA[15:0]` word for exactly four clock cycles.

For partial reconfiguration with the PR Controller IP core, specify enhanced compression by turning on the **Enhanced compression** option when specifying the parameters in the IP Catalog or Platform Designer parameter editors.

Note: You cannot use encryption with enhanced compression simultaneously.

Table 5. Partial Reconfiguration Clock Requirements for Bitstream Compression

Timing Parameters	Value (clock cycles)
`PR_READY` to first data	4 (exact)
`PR_ERROR` to last clock	80 (minimum)
`PR_DONE` to last clock	80 (minimum)
`DONE_to_REQ_low`	8 (maximum)

Related Information

- Enable Partial Reconfiguration Bitstream Decompression when Configuring Base Design SOF file in JTAG mode on page 38
- Enable Bitstream Decryption Option on page 39
- Generate PR Programming Files with the Convert Programming Files Dialog Box on page 36

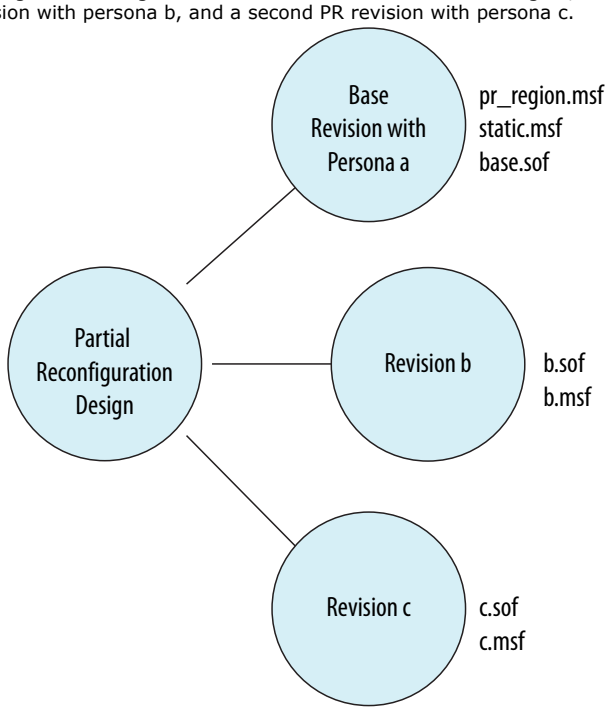
1.10. Programming Files for a Partial Reconfiguration Project

You must generate PR bitstream(s) based on the designs and send them to the control block for partial reconfiguration.

Compile the PR project, including the base revision and at least one reconfigurable revision before generating the PR bitstreams. The Intel Quartus Prime Programmer generates PR bitstreams. This generated bitstream can be sent to the PR ports on the control block for partial reconfiguration.

Figure 17. PR Project with Three Revisions

Consider a partial reconfiguration design that has three revisions and one PR region, a base revision with persona a, one PR revision with persona b, and a second PR revision with persona c.

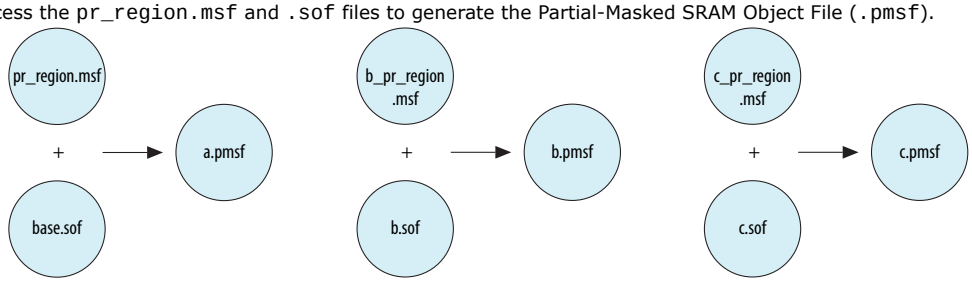


When these individual revisions are compiled in the Intel Quartus Prime software, the assembler produces Masked SRAM Object Files (`.msf`) and the SRAM Object Files (`.sof`) for each revision. The `.sof` files are created as before (for non-PR designs). Additionally, `.msf` files are created specifically for partial reconfiguration, one for each revision. The `pr_region.mfsf` file is the one of interest for generating the PR bitstream. It contains the mask bits for the PR region. Similarly, the `static.msf` file has the mask bits for the static region. The `.sof` files have the information on how to configure the static region as well as the corresponding PR region. The `pr_region.msf` file is used to mask out the static region so that the bitstream can be computed for the PR region. The default file name of the pr region `.msf` corresponds to the LogicLock region name, unless the name is not alphanumeric. In the case of a non-alphanumeric region name, the `.msf` file is named after the location of the lower left most coordinate of the region.

Note: Altera recommends naming all LogicLock regions to enhance documenting your design.

Figure 18. Generation of Partial-Masked SRAM Object Files (.pmsf)

You can convert files in the Convert Programming Files window or run the `quartus_cpf -p` command to process the `pr_region.msf` and `.sof` files to generate the Partial-Masked SRAM Object File (`.pmsf`).



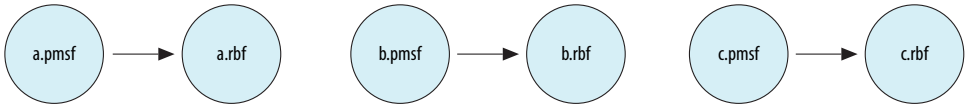
The `.msf` file helps determine the PR region from each of the `.sof` files during the PR bitstream computation.

Once all the `.pmsf` files are created, process the PR bitstreams by running the `quartus_cpf -o` command to produce the raw binary `.rbf` files for reconfiguration.

If one wishes to partially reconfigure the PR region with persona a, use the `a.rbf` bitstream file, and so on for the other personas.

Figure 19. Generating PR Bitstreams

This figure shows how three bitstreams can be created to partially reconfigure the region with persona a, persona b, or persona c as desired.



In the Intel Quartus Prime software, the Convert Programming Files window supports the generation of the required programming bitstreams. When using the `quartus_cpf` from the command line, the following options for generating the programming files are read from an option text file, for example, `option.txt`.

- If you want to use SCRUB mode, before generating the bitstreams create an option text file, with the following line:

  ```
  use_scrub=on
  ```

- If you have initialized M20K blocks in the PR region (ROM/Initialized RAM), then add the following line in the option text file, before generating the bitstreams:

  ```
  write_block_memory_contents=on
  ```

- If you want to compress the programming bitstream files, add the following line in the option text file. This option is available when converting base `.sof` to any supported programming file types, such as `.rbf`, `.pof` and JTAG Indirect Configuration File (`. jic`).

  ```
  bitstream_compression=on
  ```

Related Information

Generate PR Programming Files with the Convert Programming Files Dialog Box on page 36

1.10.1. Generating Required Programming Files

1. Generate `.sof` and `.msf` files (part of a full compilation of the base and PR revisions).
2. Generate a Partial-Masked SRAM Object File (`.pmsf`) using the following commands:

   ```
   quartus_cpf -p <pr_revision>.msf <pr_revision>.sof
   <new_filename>.pmsf
   ```

   for example:

   ```
   quartus_cpf -p x7y48.msf switchPRBS.sof x7y48_new.pmsf
   ```
3. Convert the `.pmsf` file for every PR region in your design to `.rbf` file format. The `.rbf` format is used to store the bitstream in an external flash memory. This command should be run in the same directory where the files are located:

   ```
   quartus_cpf -o scrub.txt -c <pr_revision >.pmsf
   <pr_revision>.rbf
   ```

   for example:

   ```
   quartus_cpf -o scrub.txt -c x7y48_new.pmsf x7y48.rbf
   ```

When you do not have an option text file such as `scrub.txt`, the files generated would be for AND/OR mode of PR, rather than SCRUB mode.

1.10.2. Generate PR Programming Files with the Convert Programming Files Dialog Box

In the Intel Quartus Prime software, the flow to generate PR programming files is supported in the Convert Programming Files dialog box. You can specify how the Intel Quartus Prime software processes file types such as `.msf`, `.pmsf`, and `.sof` to create `.rbf` and merged `.msf` and `.pmsf` files.

You can create

- A `.pmsf` output file, from `.msf` and `.sof` input files
- A `.rbf` output file from a `.pmsf` input file
- A merged `.msf` file from two or more `.msf` input files
- A merged `.pmsf` file from two or more `.pmsf` input files

Convert Programming Files dialog box also allows you to enable the option bit for bitstream decompression during partial reconfiguration, when converting the base `.sof` (full design `.sof`) to any supported file type.

1.10.2.1. Generating a .pmsf File from a .msf and .sof Input File

Perform the following steps in the Intel Quartus Prime software to generate the `.pmsf` file in the **Convert Programming Files** dialog box.

1. Open the **Convert Programming Files** dialog box.
2. Specify the programming file type as `Partial-Masked SRAM Object File (.pmsf)`.
3. Specify the output file name.
4. Select input files to convert (only a single `.msf` and `.sof` file are allowed). Click **Add**.
5. Click **Generate** to generate the `.pmsf` file.

1.10.2.2. Generating a .rbf File from a .pmsf Input File

Perform the following steps in the Intel Quartus Prime software to generate the partial reconfiguration `.rbf` file in the **Convert Programming Files** dialog box.

1. From the File menu, click **Convert Programming Files**.
2. Specify the programming file type as `Raw Binary File for Partial Reconfiguration (.rbf)`.
3. Specify the output file name.
4. Select input file to convert. Only a single `.pmsf` input file is allowed. Click **Add**.
5. Select the new `.pmsf` and click **Properties**.
6. Turn the **Compression**, **Enable SCRUB mode**, **Write memory contents**, and **Generate encrypted bitstream** options on or off depending on the requirements of your design. Click **Generate** to generate the `.rbf` file for partial reconfiguration.

- **Compression**: Enables compression on the PR bitstream.
- **Enable SCRUB mode**: Default is based on AND/OR mode. This option is valid only when your design does not contain vertically overlapped PR masks. The `.rbf` generation fails otherwise.
- **Write memory contents**: Turn this on when you have a `.mif` that was used during compilation. Otherwise, turning this option on forces you to use double PR in AND/OR mode.
- **Generate encrypted bitstream**: If this option is enabled, you must specify the Encrypted Key Programming (`.ekp`) file, which generated when converting a base `.sof` to an encrypted bitstream. The same `.ekp` must be used to encrypt the PR bitstream.

When you turn on **Compression**, you must present each `PR_DATA[15:0]` word for exactly four clock cycles.

Turn on the **Write memory contents** option only if you are using AND/OR mode and have M20K blocks in your PR design that need to be initialized. When you check this box, you must to perform double PR for regions with initialized M20K blocks.

Related Information

Initializing M20K Blocks with a Double PR Cycle on page 45

1.10.2.3. Create a Merged .msf File from Multiple .msf Files

You can merge two or more `.msf` files in the **Convert Programming Files** window.

1. Open the **Convert Programming Files** window.
2. Specify the programming file type as `Merged Mask Settings File (.msf)`.
3. Specify the output file name.
4. Select **MSF Data** in the **Input files to convert** window.
5. Click Add File to add input files. You must specify two or more files for merging.
6. Click **Generate**to generate the merged file.

To merge two or more `.msf` files from the command line, type:

```
quartus_cpf --merge_msf=<number of merged files>
<msf_input_file_1> <msf_input_file_2> <msf_input_file_etc>
<msf_output_file>
```

For example, to merge two `.msf` files, type:

```
quartus_cpf --merge_msf=<2> <msf_input_file_1> <msf_input_file_2>
<msf_output_file>
```

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

You can merge two or more `.pmsf` files in the **Convert Programming Files** window.

1. Open the **Convert Programming Files** window.
2. Specify the programming file type as `Merged Partial-Mask SRAM Object File (.pmsf)`.
3. Specify the output file name.
4. Select **PMSF Data** in the **Input files to convert** window.
5. Click **Add File** to add input files. You must specify two or more files for merging.
6. Click **Generate** to generate the merged file.

To merge two or more `.pmsf` files from the command line, type:

```
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:

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

The merge operation checks for any bit conflict on the input files, and the operation fails with error message if a bit conflict is detected. In most cases, a successful file merge operation indicates input files do not have any bit conflict.

1.10.2.5. Enable Partial Reconfiguration Bitstream Decompression when Configuring Base Design SOF file in JTAG mode

In the Intel Quartus Prime software, the **Convert Programming Files** window provides the option in the `.sof` file properties to enable bitstream decompression during partial reconfiguration.

This option is available when converting base `.sof` to any supported programming file types, such as `.rbf`, `.pof`, and `.jic`.

In order to view this option, the base `.sof` must be targeted on Stratix V devices in the `.sof` **File Properties**. This option must be turned on if you turned on the **Compression** option during `.pmsf` to `.rbf` file generation.

1.10.2.6. Enable Bitstream Decryption Option

The **Convert Programming Files** window provides the option in the `.sof` file properties to enable bitstream decryption during partial reconfiguration.

This option is available when converting base `.sof` to any supported programming file types, such as `.rbf`, `.pof`, and `.jic`.

The base `.sof` must have partial reconfiguration enabled and the base `.sof` generated from a design that has a PR Control Block instantiated, to view this option in the `.sof` **File Properties**. This option must be turned on if you wants to turn on the Generate encrypted bitstream option during `.pmsf` to `.rbf` file generation.

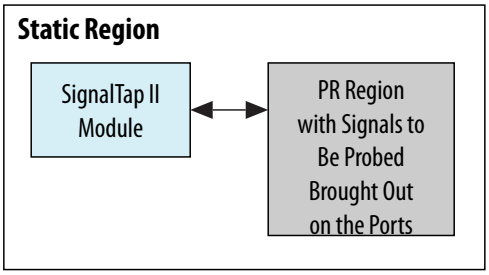
1.11. On-Chip Debug for PR Designs

You cannot instantiate a Signal Tap block inside a PR region. If you must monitor signals within a PR region for debug purposes, bring those signals to the ports of the PR region.

The Intel Quartus Prime software does not support the Incremental Signal Tap feature for PR designs. After you instantiate the Signal Tap block inside the static region, you must recompile your design. When you recompile your design, the static region may have a modified implementation and you must also recompile your PR revisions. If you modify an existing Signal Tap instance you must also recompile your entire design; base revision and reconfigurable revisions.

Figure 20. Using Signal Tap with a PR Design

You can instantiate the SignalTap II block in the static region of the design and probe the signals you want to monitor.



You can use other on-chip debug features in the Intel Quartus Prime software, such as the In-System Sources and Probes or Signal Probe, to debug a PR design. As in the case of SignalTap, In-System Sources and Probes can only be instantiated within the static region of a PR design. If you have to probe any signal inside the PR region, you must bring those signals to the ports of the PR region in order to monitor them within the static region of the design.

1.12. Partial Reconfiguration Known Limitations

There are restrictions that derive from hardware limitations in specific Stratix V devices.

The restrictions in the following sections apply only if your design uses M20K blocks as RAMs or ROMs in your PR project.

1.12.1. Memory Blocks Initialization Requirement for PR Designs

For a non-PR design, the power up value for the contents of a M20K RAM or a MLAB RAM are all set at zero. However, at the end of performing a partial reconfiguration, the contents of a M20K or MLAB memory block are unknown. You must intentionally initialize the contents of all the memory to zero, if required by the functionality of the design, and not rely upon the power on values.

1.12.2. M20K RAM Blocks in PR Designs

When your PR design uses M20K RAM blocks in Stratix V devices, there are some restrictions which limit how you utilize the respective memory blocks as ROMs or as RAMs with initial content.

Related Information

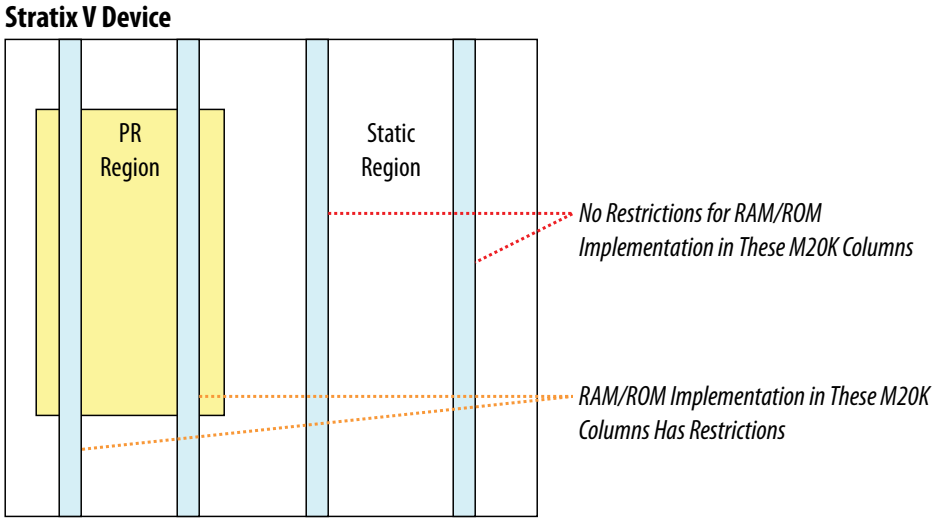
Implementing Memories with Initialized Content on page 43
If your design requires initialized memory content either as a ROM or a RAM inside a PR region, you must follow these guidelines.

1.12.2.1. Limitations When Using Stratix V Production Devices

These workarounds allow your design to use M20K blocks with PR.

Figure 21. Limitations for Using M20Ks in PR Regions

If you implement a M20K block in your PR region as a ROM or a RAM with initialized content, when the PR region is reconfigured, any data read from the memory blocks in static regions in columns that cross the PR region is incorrect.



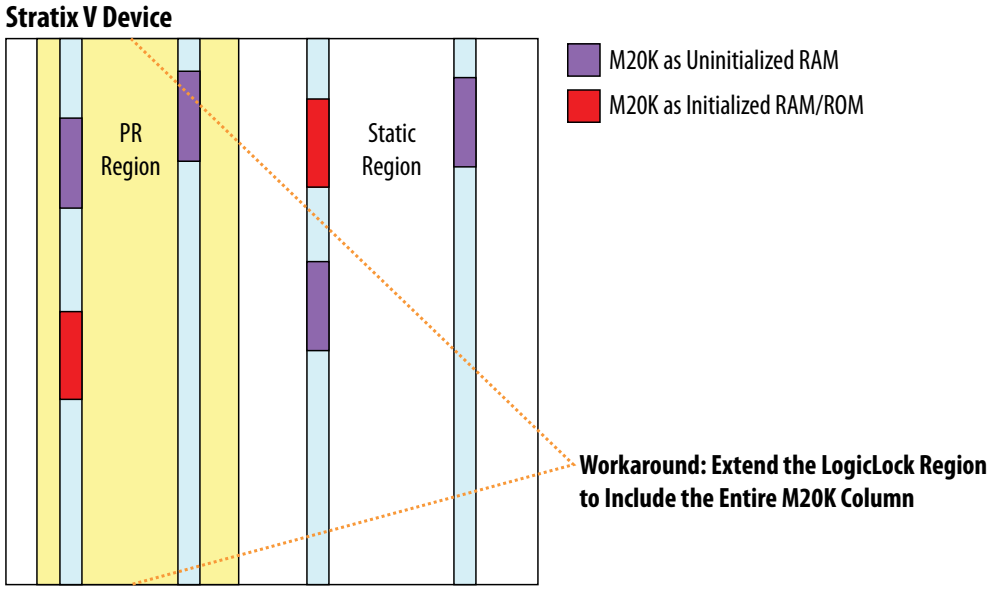
If the functionality of the static region depends on any data read out from M20K RAMs in the static region, the design will malfunction.

Use one of the following workarounds, which are applicable to both AND/OR and SCRUB modes of partial reconfiguration:

- Do not use ROMs or RAMs with initialized content inside PR regions.
- If this is not possible for your design, you can program the memory content for M20K blocks with a `.mif` using the suggested workarounds.
- Make sure your PR region extends vertically all the way through the device, in such a way that the M20K column lies entirely inside a PR region.

Figure 22. Workaround for Using M20Ks in PR Regions

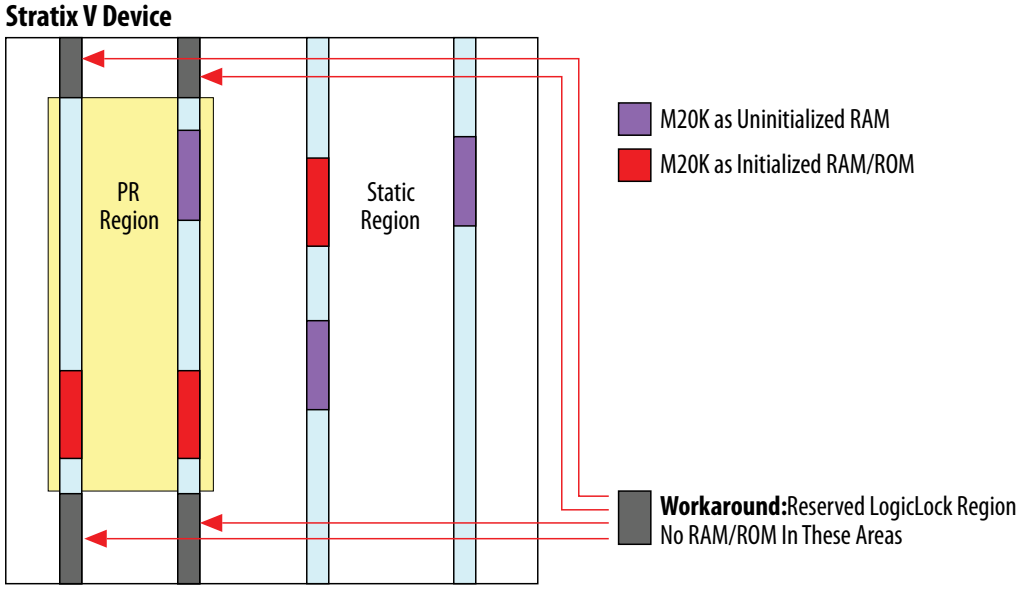
This figure shows the LogicLock region extended as a rectangle reducing the area available for the static region. However, you can create non-rectangular LogicLock regions to allocate the resources required for the partition more optimally. If saving area is a concern, extend the LogicLock region to include M20K columns entirely.



-

Figure 23. Alternative Workaround for Using M20Ks in PR Region

Using Reserved LogicLock Regions, block all the M20K columns that are not inside a PR region, but that are in columns above or below a PR region. In this case, you may choose to under-utilize M20K resources, in order to gain ROM functionality within the PR region.



For more information including a list of the Stratix V production devices, refer to the *Errata Sheet for Stratix V Devices*.

1.12.3. MLAB Blocks in PR designs

Stratix V devices include dual-purpose blocks called MLABs, which can be used to implement RAMs or LABs for user logic.

This section describes the restrictions while using MLAB blocks (sometimes also referred to as LUT-RAM) in Stratix V devices for your PR designs.

If your design uses MLABS as LUT RAM, you must use all available MLAB bits within the region.

Table 6. RAM Implementation Restrictions Summary

The following table shows a summary of the LUT-RAM Restrictions.

PR Mode	Type of memory in PR region	Stratix V Production
SCRUB mode	LUT RAM (no initial content)	OK
	LUT ROM and LUT RAM with your initial content	OK
AND/OR mode	LUT RAM (no initial content)	*While design is running:* Write 1s to all locations before partial reconfiguration *At compile time:* Explicitly initialize all memory locations in each new persona to 1 via initialization file (**. mif**).
	LUT ROM and LUT RAM with your initial content	No

If your design does not use any MLAB blocks as RAMs, the following discussion does not apply. The restrictions listed below are the result of hardware limitations in specific devices.

Limitations with Stratix V Production Devices

When using SCRUB mode:

- LUT-RAMs without initialized content, LUT-RAMs with initialized content, and LUT-ROMs can be implemented in MLABs within PR regions without any restriction.

When using AND/OR mode:

- LUT-RAMs with initialized content or LUT-ROMs cannot be implemented in a PR region.
- LUT-RAMs without initialized content in MLABs inside PR regions are supported with the following restrictions.
- MLAB blocks contain 640 bits of memory. The LUT RAMs in PR regions in your design must occupy all MLAB bits, you should not use partial MLABs.
- You must include control logic in your design with which you can write to all MLAB locations used inside PR region.
- Using this control logic, write '1' at each MLAB RAM bit location in the PR region before starting the PR process. This is to work around a false EDCRC error during partial reconfiguration.
- You must also specify a `.mif` that sets all MLAB RAM bits to '1' immediately after PR is complete.
- ROMs cannot be implemented in MLABs (LUT-ROMs).
- There are no restrictions to using MLABs in the static region of your PR design.

For more information, refer to the following documents in the *Stratix V Handbook*:

1.12.4. Implementing Memories with Initialized Content

If your Stratix V PR design implements ROMs, RAMs with initialization, or ROMs within the PR regions, using either M20K blocks or LUT-RAMs, then you must follow the following design guidelines to determine what is applicable in your case.

Table 7. Implementing Memory with Initialized Content in PR Designs

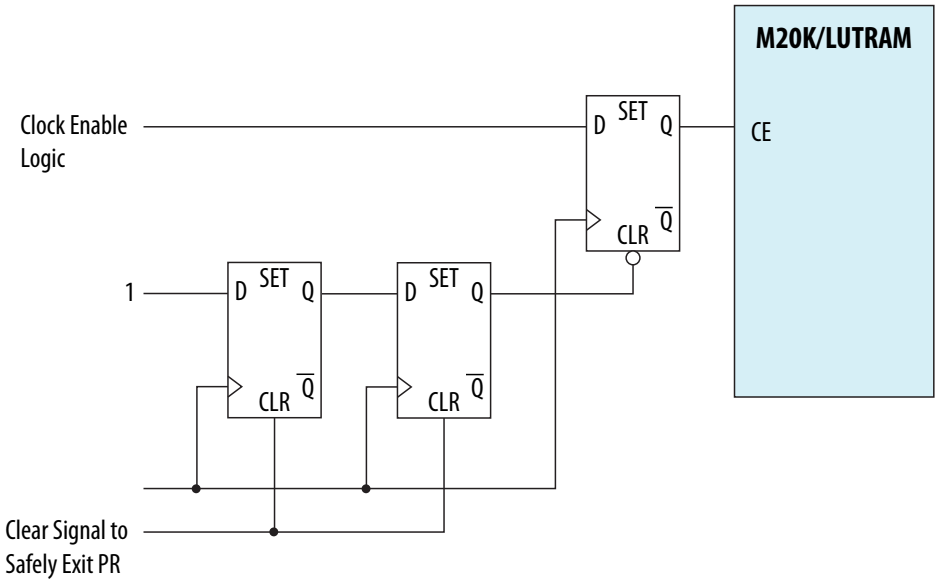
Mode		Production Devices	
		AND/OR	SCRUB
LUT-RAM without initialization	**Suggested Method**	*While design is running:* Write '1' to all locations before partial reconfiguration. *At compile time:* Explicitly initialize all memory locations in each new persona to '1' via initialization file (`.mif`)	No special method required

continued...

Mode		Production Devices	
		AND/OR	SCRUB
		Make sure no spurious write on PR entry (1)	
	Without Suggested Method	CRC Error	No special method required
LUT-RAM with initialization	**Suggested Method**	Not supported	Make sure no spurious write on PR exit (1)
	Without Suggested Method		Incorrect results
M20K without initialization	**Suggested Method**	No special method required	
	Without Suggested Method	No special method required	
M20K with initialization	**Suggested Method**	Use double PR cycle (2) Make sure no spurious write on PR exit (1)	No special method required
	Without Suggested Method	Incorrect results	No special method required

Figure 24. M20K/LUTRAM

To avoid spurious writes during PR entry and exit, implement the following clock enable circuit in the same PR region as the RAM.



The circuit depends on an active- high clear signal from the static region. Before entering PR, freeze this signal in the same manner as all PR inputs. Your host control logic should de-assert the clear signal as the final step in the PR process.

(1) Use the circuit shown in the **M20K/LUTRAM** figure to create clock enable logic to safely exit partial reconfiguration without spurious writes.

(2) Double partial reconfiguration is described in *Initializing M20K Blocks with a Double PR Cycle*

Related Information

Initializing M20K Blocks with a Double PR Cycle on page 45

1.12.5. Initializing M20K Blocks with a Double PR Cycle

When a PR region in your PR design contains an initialized M20K block and is reconfigured via AND/OR mode, your host logic must complete a double PR cycle, instead of a single PR cycle.

The PR IP has a `double_pr` input port, that must be asserted high when your PR region contains RAM blocks that must be initialized. The PR IP core handles the timing relations between the first and the second PR cycles of a Double PR operation. From your user logic, assert the `double_pr` signal when you assert the `pr_start` signal, and you deassert the `double_pr` signal when the `freeze` signal is deasserted by the PR IP. This method is also applicable in cases when the programming bitstream is compressed or encryted.

1.13. Document Revision History

Table 8. Document Revision History

Date	Version	Changes
2017.11.06	17.1.0	• Updated PR Bitstream Compression and Encryption topic to clarify FPGA family differences.
2015.11.02	15.1.0	Changed instances of *Quartus II* to *Intel Quartus Prime*.
2015.05.04	15.0.0	• Correct Verilog HDL partial reconfiguration instantiation code example. • Added clear/set method to SCRUB mode option.
2015.12.15	14.1.0	Minor revisions to some topics to resolve design refinements: • Implementing Memories with Initialized Content • Instantiating the PR Control Block and CRC Block in Verilog HDL • Partial Reconfiguration Pins
June 2014	14.0.0	Minor updates to "Programming File Sizes for a Partial Reconfiguration Project" and code samples in "Freeze Logic for PR Regions" sections.
November 2013	13.1.0	Added support for merging multiple .msf and .pmsf files. Added support for PR Megafunction. Updated for revisions on timing requirements.
May 2013	13.0.0	Added support for encrypted bitstreams. Updated support for double PR.
November 2012	12.1.0	Initial release.

Related Information

Documentation Archive

For previous versions of the *Intel Quartus Prime Handbook*, search the documentation archives.

A. Intel Quartus Prime Standard Edition User Guides

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

Related Information

- Intel Quartus Prime Standard Edition User Guide: Getting Started
  Introduces the basic features, files, and design flow of the Intel Quartus Prime Standard Edition software, including managing Intel Quartus Prime Standard Edition projects and IP, initial design planning considerations, and project migration from previous software versions.
- Intel Quartus Prime Standard Edition User Guide: Platform Designer
  Describes creating and optimizing systems using Platform Designer (Standard), a system integration tool that simplifies integrating customized IP cores in your project. Platform Designer (Standard) automatically generates interconnect logic to connect intellectual property (IP) functions and subsystems.
- Intel Quartus Prime Standard Edition User Guide: Design Recommendations
  Describes best design practices for designing FPGAs with the Intel Quartus Prime Standard Edition software. HDL coding styles and synchronous design practices can significantly impact design performance. Following recommended HDL coding styles ensures that Intel Quartus Prime Standard Edition synthesis optimally implements your design in hardware.
- Intel Quartus Prime Standard Edition User Guide: Design Compilation
  Describes set up, running, and optimization for all stages of the Intel Quartus Prime Standard Edition Compiler. The Compiler synthesizes, places, and routes your design before generating a device programming file.
- Intel Quartus Prime Standard Edition User Guide: Design Optimization
  Describes Intel Quartus Prime Standard 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, and optimization of device resource usage.
- Intel Quartus Prime Standard Edition User Guide: Programmer
  Describes operation of the Intel Quartus Prime Standard 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 Standard Edition User Guide: Partial Reconfiguration
  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 Standard 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.
- Intel Quartus Prime Standard Edition User Guide: Third-party Synthesis
  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.
- Intel Quartus Prime Standard Edition User Guide: Debug Tools
  Describes a portfolio of Intel Quartus Prime Standard 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, Transceiver Toolkit, In-System Memory Content Editor, and In-System Sources and Probes Editor.
- Intel Quartus Prime Standard Edition User Guide: Timing Analyzer
  Explains basic static timing analysis principals and use of the Intel Quartus Prime Standard 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.
- Intel Quartus Prime Standard Edition User Guide: Power Analysis and Optimization
  Describes the Intel Quartus Prime Standard 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 Standard 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 Pin Planner to visualize, modify, and validate all I/O assignments in a graphical representation of the target device.
- Intel Quartus Prime Standard Edition User Guide: PCB Design Tools
  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 Standard Edition User Guide: Scripting
  Describes use of Tcl and command line scripts to control the Intel Quartus Prime Standard Edition software and to perform a wide range of functions, such as managing projects, specifying constraints, running compilation or timing analysis, or generating reports.