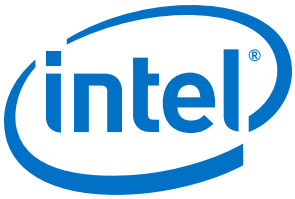


AN 806: Hierarchical Partial Reconfiguration Tutorial for Intel® Arria® 10 GX FPGA Development Board

Updated for Intel® Quartus® Prime Design Suite: **18.0**

AN-806 | 2018.05.07
Latest document on the web: **PDF** | **HTML**

Contents

Hierarchical Partial Reconfiguration Tutorial for Arria® 10 GX FPGA Development Board

This application note demonstrates transforming a simple design into a hierarchically partially reconfigurable design, and implementing the design on the Arria® 10 GX FPGA development board.

Hierarchical partial reconfiguration (HPR) is an extension of the traditional partial reconfiguration (PR), where you contain a PR region within another PR region. You can create multiple personas for both the child and parent partitions. You nest the child partitions within their parent partitions. Reconfiguring a parent partition does not impact the operation in the static region, but replaces the child partitions of the parent region with default child partition personas. This methodology is effective in systems where multiple functions time-share the same FPGA device resources.

Partial reconfiguration provides the following advancements to a flat design:

- Allows run-time design reconfiguration
- Increases scalability of the design
- Reduces system down-time
- Supports dynamic time-multiplexing functions in the design
- Lowers cost and power consumption through efficient use of board space

The current version of the Intel® Quartus® Prime Pro Edition software introduces a new and simplified compilation flow for partial reconfiguration.

Implementation of this reference design requires basic familiarity with the Intel Quartus Prime FPGA implementation flow and knowledge of the primary Intel Quartus Prime project files. This tutorial uses the Arria 10 GX FPGA development board on the bench, outside of the PCIe* slot in your workstation.

Related Information

- Intel Arria 10 FPGA Development Kit User Guide
- Partial Reconfiguration Concepts
- Partial Reconfiguration Design Flow
- Partial Reconfiguration Design Considerations
- Partial Reconfiguration Design Guidelines

Reference Design Requirements

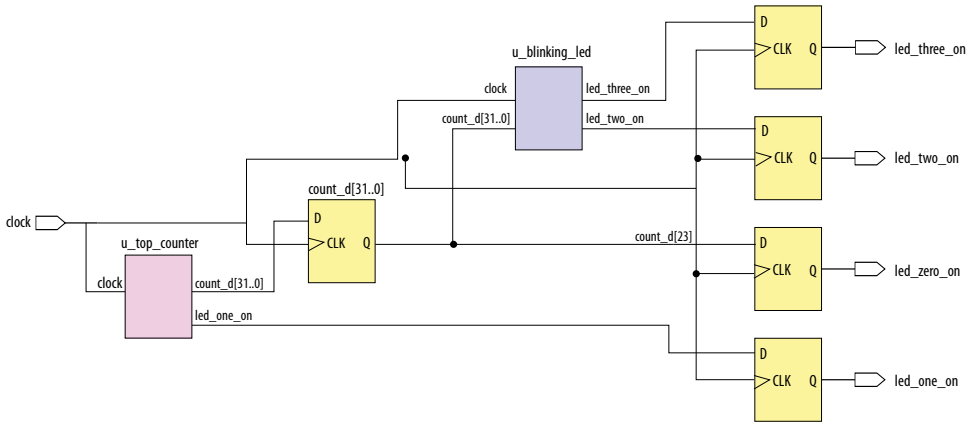
This reference design requires the following:

- Intel Quartus Prime Pro Edition software version 18.0 for the design implementation.
- Arria 10 GX FPGA development kit for the FPGA implementation.

Reference Design Overview

This reference design consists of one 32-bit counter. At the board level, the design connects the clock to a 50MHz source, and connects the output to four LEDs on the FPGA. Selecting the output from the counter bits in a specific sequence causes the LEDs to blink at a specific frequency.

Figure 1. Flat Reference Design without PR Partitioning



Reference Design Files

The partial reconfiguration tutorial is available in the following location:

https://github.com/intel/fpga-partial-reconfig

To download the tutorial:

1. Click **Clone or download**.
2. Click **Download ZIP**. Unzip the `fpga-partial-reconfig-master.zip` file.
3. Navigate to the `tutorials/a10_pcie_devkit_blinking_led_hpr` sub-folder to access the reference design.

The `flat` folder consists of the following files:

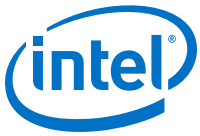
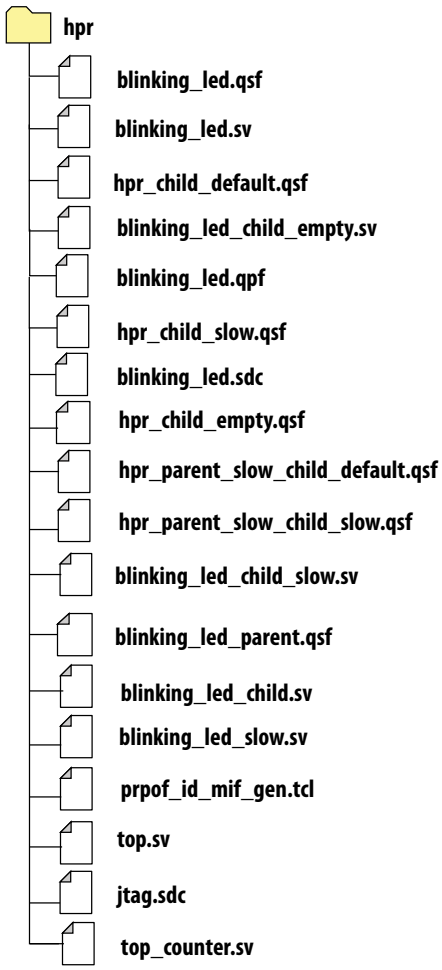


Table 1. Reference Design Files

File Name	Description
`top.sv`	Top-level file containing the flat implementation of the design. This module instantiates the `blinking_led` sub-partition and the `top_counter` module.
`top_counter.sv`	Top-level 32-bit counter that controls `LED[1]` directly. The registered output of the counter controls `LED[0]`, and also powers `LED[2]` and `LED[3]` via the `blinking_led` module.
`blinking_led.sdc`	Defines the timing constraints for the project.
`blinking_led.sv`	In this tutorial, you convert this module into a parent PR partition. The module receives the registered output of `top_counter` module, which controls `LED[2]` and `LED[3]`.
`blinking_led.qpf`	Intel Quartus Prime project file containing the list of all the revisions in the project.
`blinking_led.qsf`	Intel Quartus Prime settings file containing the assignments and settings for the project.
`prpof_id_mif_gen.tcl`	Script file to enable bitstream compatibility checks for child PR regions.

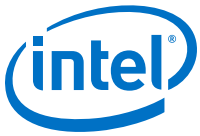
Note: The `hpr` folder contains the complete set of files you create using this application note. Reference these files at any point during the walkthrough.

Figure 2. Reference Design Files



Reference Design Walkthrough

The following steps describe the application of partial reconfiguration to a flat design. The tutorial uses the Intel Quartus Prime Pro Edition software for the Arria 10 GX FPGA development board:



Step 1: Getting Started

To copy the reference design files to your working environment and compile the `blinking_led` flat design:

1. Create a directory in your working environment, `a10_pcie_devkit_blinking_led_hpr`.
2. Copy the downloaded `tutorials/a10_pcie_devkit_blinking_led_hpr/flat` sub-folder to the directory, `a10_pcie_devkit_blinking_led_hpr`.
3. In the Intel Quartus Prime Pro Edition software, click **File ➤ Open Project** and select `blinking_led.qpf`.
4. To compile the flat design, click **Processing ➤ Start Compilation**.

Step 2: Creating a Child Level Sub-module

To convert this flat design into a hierarchical PR design, you must create a child sub-module (`blinking_led_child.sv`) that is nested within the parent sub-module (`blinking_led.sv`).

1. Create a new design file, `blinking_led_child.sv`, and add the following lines of code to this file:

```
`timescale 1 ps / 1 ps
`default_nettype none

module blinking_led_child (

   // clock
   input wire clock,
   input wire [31:0] counter,


   // Control signals for the LEDs
   output wire led_three_on

);
   localparam COUNTER_TAP = 23;
   reg led_three_on_r;


   assign led_three_on   = led_three_on_r;

   always_ff @(posedge clock) begin
      led_three_on_r   <= counter[COUNTER_TAP];
```

```
   end

endmodule
```

2. Modify the `blinking_led.sv` file to connect the `led_two_on` to bit 23 of the counter from the static region, and instantiate the `blinking_led_child` module. After modifications, your `blinking_led.sv` file must appear as follows:

```
`timescale 1 ps / 1 ps
`default_nettype none

module blinking_led(
   // clock
   input wire clock,
   input wire [31:0] counter,
   // Control signals for the LEDs
   output wire led_two_on,
   output wire led_three_on
);


   localparam COUNTER_TAP = 23;

   reg led_two_on_r;
   assign  led_two_on    = led_two_on_r;

   // The counter:
   always_ff @(posedge clock) begin
         led_two_on_r <= counter[COUNTER_TAP];
   end


   blinking_led_child u_blinking_led_child (
         .led_three_on           (led_three_on),
         .counter                (counter),
         .clock                  (clock)
   );

endmodule
```

3. On modifying all the design files, recompile the project by clicking **Processing ➤ Start Compilation**

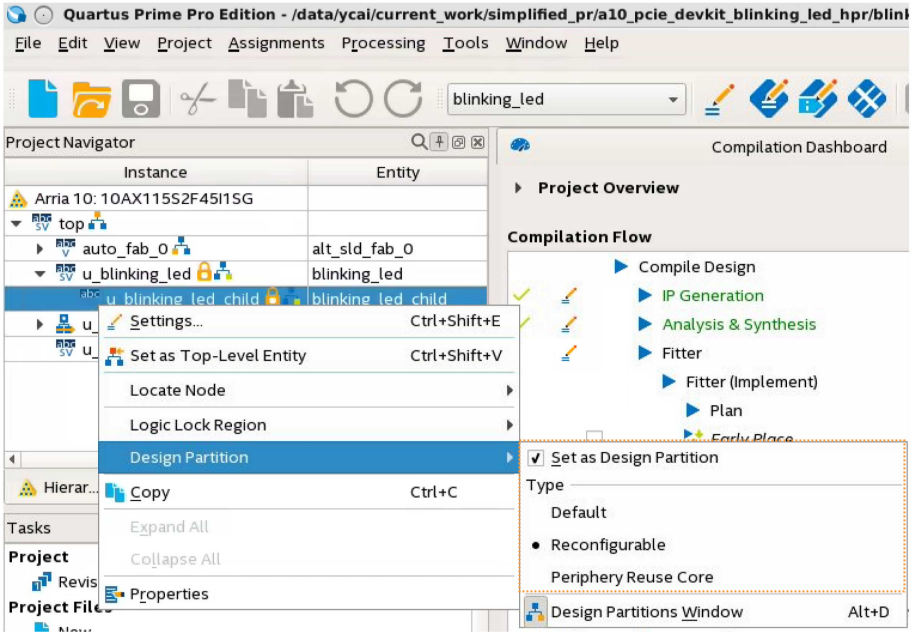
Step 3: Creating Design Partitions

You must create design partitions for each PR region that you want to partially reconfigure. You can create any number of independent partitions or PR regions in your design. This tutorial creates two design partitions for the `u_blinking_led_child` and `u_blinking_led` instances.

To create design partitions for hierarchical partial reconfiguration:

1. Right-click the `u_blinking_led_child` instance in the **Project Navigator** and click **Design Partition ➤ Set as Design Partition**. A design partition icon appears next to each instance that is set as a partition.

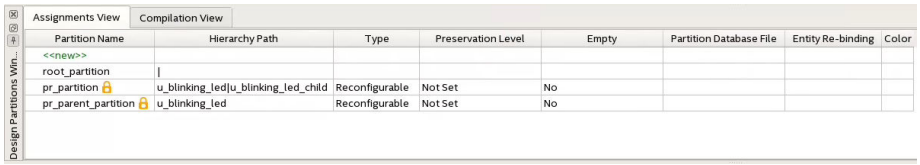
Figure 3. Creating Design Partitions from Project Navigator



2. To define the partition **Type**, right-click the `u_blinking_led_child` instance in the **Hierarchy** tab, click **Design Partition ➤ Reconfigurable**. You can only define the partition **Type** after setting the instance as a partition.

   The design partition appears on the **Assignments View** tab of the Design Partitions Window.

Figure 4. Design Partitions Window



3. Edit the partition name in the Design Partitions Window by double-clicking the name. For this reference design, rename the partition name to `pr_partition`.

   Note: When you create a partition, the Intel Quartus Prime software automatically generates a partition name, based on the instance name and hierarchy path. This default partition name can vary with each instance.

Verify that the `blinking_led.qsf` contains the following assignments, corresponding to your reconfigurable design partitions:

```
set_instance_assignment -name PARTITION pr_partition -to \
        u_blinking_led|u_blinking_led_child
set_instance_assignment -name PARTIAL_RECONFIGURATION_PARTITION ON -to \
        u_blinking_led|u_blinking_led_child

set_instance_assignment -name PARTITION pr_parent_partition -to u_blinking_led
set_instance_assignment -name PARTIAL_RECONFIGURATION_PARTITION ON -to \
        u_blinking_led
```

Related Information

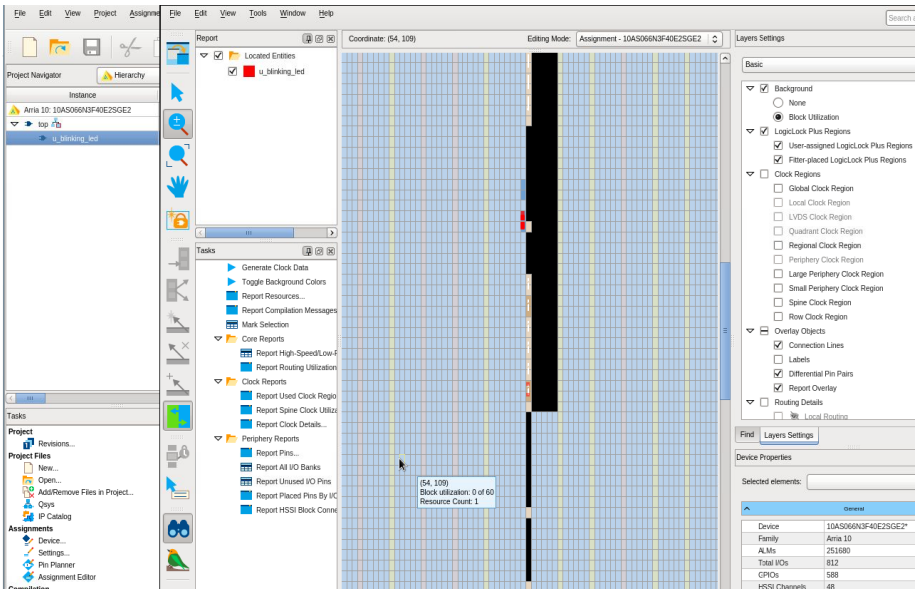
Create Design Partitions for Partial Reconfiguration

Step 4: Allocating Placement and Routing Region for PR Partitions

When you create the base revision, the PR design flow uses your PR partition region allocation to place the corresponding persona core in the reserved region. To locate and assign the PR region in the device floorplan for your base revision:

1. Right-click the `u_blinking_led_child` instance in the **Project Navigator** and click **Logic Lock Region ➤ Create New Logic Lock Region**. The region appears on the Logic Lock Regions Window.
2. Your placement region must enclose the `blinking_led_child` logic. Select the placement region by locating the node in Chip Planner. Right-click the `u_blinking_led_child` region name in the **Project Navigator** and click **Locate Node ➤ Locate in Chip Planner**.

Figure 5. Chip Planner Node Location for `blinking_led`



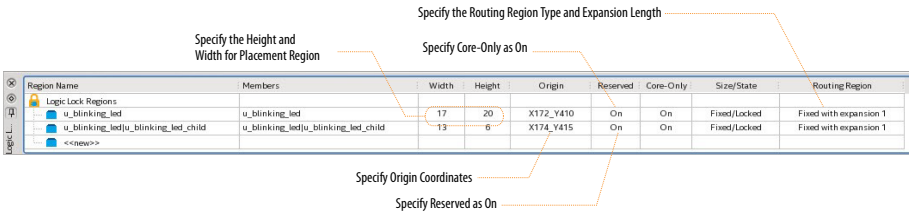
3. In the Logic Lock Regions window, specify the placement region co-ordinates in the **Origin** column. The origin corresponds to the lower-left corner of the region. For example, to set a placement region with (X1 Y1) co-ordinates as (69 10), specify the **Origin** as X69_Y10. The Intel Quartus Prime software automatically calculates the (X2 Y2) co-ordinates (top-right) for the placement region, based on the height and width you specify.

   Note: This tutorial uses the (X1 Y1) co-ordinates - (69 10), and a height and width of 20 for the placement region. Define any value for the placement region, provided that the region covers the `blinking_led_child` logic.
4. Enable the **Reserved** and **Core-Only** options.

5. Double-click the **Routing Region** option. The **Logic Lock Routing Region Settings** dialog box appears.
6. Select **Fixed with expansion** for the **Routing type**. Selecting this option automatically assigns an expansion length of 1.

   Note: The routing region must be larger than the placement region, to provide extra flexibility for the Fitter when the engine routes different personas.
7. Repeat steps 1 -6 for the `u_blinking_led` instance. The parent-level placement region must fully enclose the corresponding child-level placement and routing regions, while allowing sufficient space for the parent-level logic placement. This tutorial uses the (X1 Y1) co-ordinates - (66 7), a height of 47, and width of 26 for the placement region of the `u_blinking_led` instance.

Figure 6. Logic Lock Regions Window



Verify that the `blinking_led.qsf` contains the following assignments, corresponding to your floorplanning:

```
set_instance_assignment -name PLACE_REGION "69 10 88 29" -to \
        u_blinking_led|u_blinking_led_child
set_instance_assignment -name RESERVE_PLACE_REGION ON -to \
        u_blinking_led|u_blinking_led_child
set_instance_assignment -name CORE_ONLY_PLACE_REGION ON -to \
        u_blinking_led|u_blinking_led_child
set_instance_assignment -name ROUTE_REGION "68 9 89 30" -to \
        u_blinking_led|u_blinking_led_child

set_instance_assignment -name PLACE_REGION "66 7 112 32" -to u_blinking_led
set_instance_assignment -name RESERVE_PLACE_REGION ON -to u_blinking_led
set_instance_assignment -name CORE_ONLY_PLACE_REGION ON -to u_blinking_led
set_instance_assignment -name ROUTE_REGION "65 6 113 33" -to u_blinking_led
```

Related Information

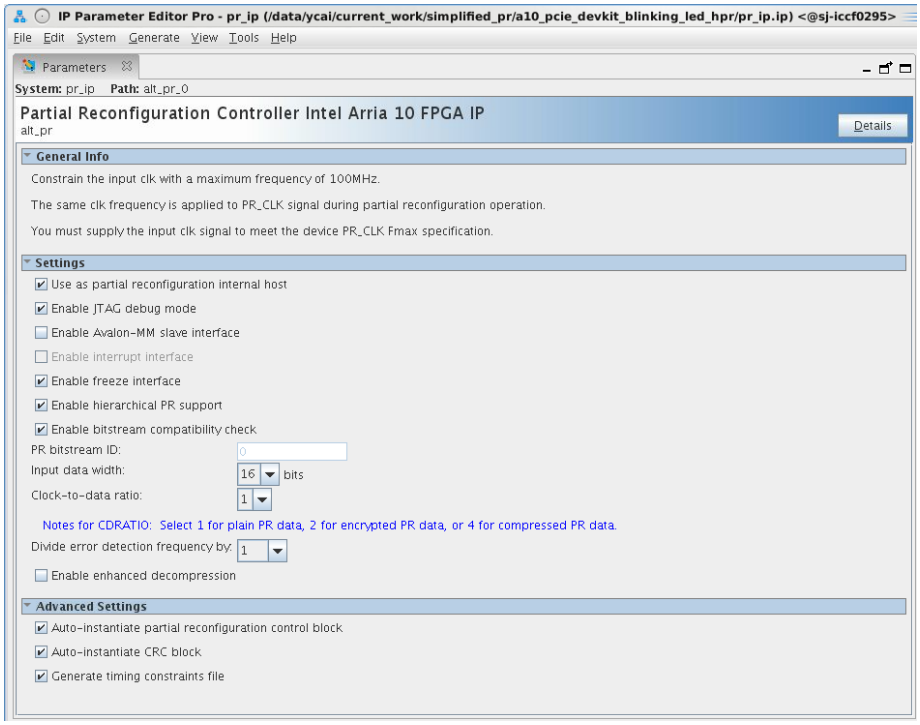
- Floorplan the Partial Reconfiguration Design
- Applying Floorplan Constraints Incrementally

Step 5: Adding the Arria 10 Partial Reconfiguration Controller IP Core

Use the Arria 10 Partial Reconfiguration Controller IP core to reconfigure the PR partition. This IP core uses JTAG to reconfigure the PR partition. To add the Arria 10 Partial Reconfiguration Controller IP core to your Intel Quartus Prime project:

1. Type `Partial Reconfiguration` in the IP catalog.
2. To launch the IP Parameter Editor Pro window, select the Arria 10 Partial Reconfiguration Controller IP core from the IP library, and click **Add**.
3. In the **New IP Variant** dialog box, type `pr_ip` as the file name and click **Create**. Use the default parameterization for `pr_ip`. Ensure that the **Enable JTAG debug mode** and **Enable freeze interface** options are turned on, **Enable Avalon-MM slave interface** option is turned off, **Enable hierarchical PR support** option is turned on, and **Enable bitstream compatibility check** option is turned on .

Figure 7. Arria 10 Partial Reconfiguration Controller IP Core Parameters



4. Click **Finish**, and exit the parameter editor without generating the system. Intel Quartus Prime software creates the `pr_ip.ip` IP variation file, and adds the file to the `blinking_led` project.

Note:

1. If you are copying the `pr_ip.ip` file from the `hpr` folder, manually edit the `blinking_led.qsf` file to include the following line:

```
set_global_assignment -name IP_FILE pr_ip.ip
```

2. Place the `IP_FILE` assignment after the `SDC_FILE` assignments (`jtag.sdc` and `blinking_led.sdc`) in your `blinking_led.qsf` file. This ordering ensures appropriate constraining of the Partial Reconfiguration IP core.

   Note: To detect the clocks, the SDC file for the PR IP must follow any SDC that creates the clocks that the IP core uses. You facilitate this order by ensuring the `.ip` file for the PR IP core comes after any `.ip` files or SDC files used to create these clocks in the QSF file for your Intel Quartus Prime project revision. For more information, refer to *Timing Constraints* section in the *Partial Reconfiguration IP Core User Guide*.

Related Information

- Partial Reconfiguration IP Solutions User Guide
  For information on the Partial Reconfiguration Region Controller IP core.
- Partial Reconfiguration IP Core User Guide
  For information on the timing constraints.

Updating the Top-Level Design

To update the `top.sv` file with the PR_IP instance:

1. To add the PR_IP instance to the top-level design, uncomment the following code block in `top.sv` file:

```
pr_ip u_pr_ip
    (
        .clk           (clock),
        .nreset        (1'b1),
        .freeze        (freeze),
        .pr_start      (1'b0),            // ignored for JTAG
        .status        (pr_ip_status),
        .data          (16'b0),
        .data_valid    (1'b0),
        .data_ready    ()
    );
```

2. To force the output ports to logic 1 during reconfiguration, use the freeze control signal output from PR_IP. However, to observe the LED continue blinking from the parent PR partition while PR programming the child partition, the freeze control signal does not turn off the `led_two_on`. Ensure that the `pr_led_two_on` is directly assigned to `led_two_on_w`. `led_three_on_w` must choose between logic 1 and `pr_led_three_on`, based on the freeze signal. Uncomment the following lines of code:

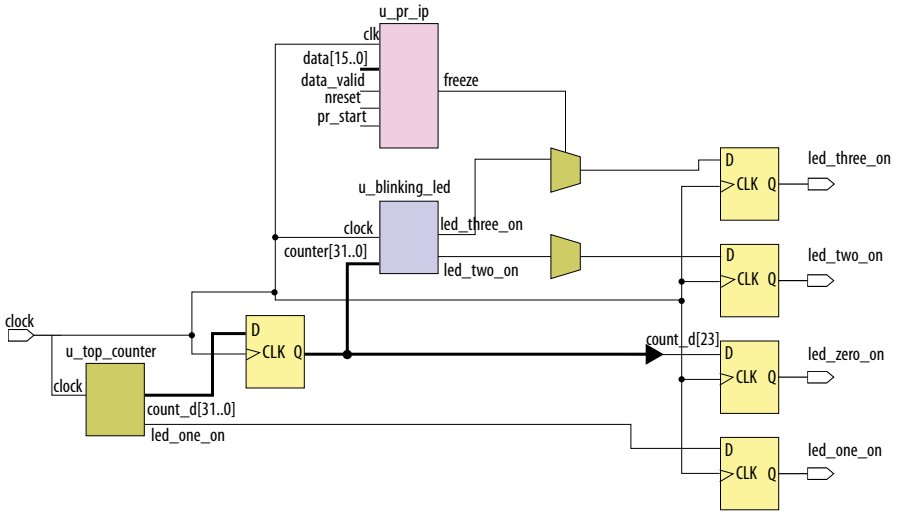
```
assign led_two_on_w = pr_led_two_on;
assign led_three_on_w = freeze ? 1'b1 : pr_led_three_on;
```

3. To assign an instance of the default parent persona (`blinking_led`), update the `top.sv` file with the following block of code:

```
blinking_led u_blinking_led
   (
      .clock         (clock),
      .counter       (count_d),
      .led_two_on    (pr_led_two_on),
      .led_three_on  (pr_led_three_on)

   );
```

Figure 8. Partial Reconfiguration IP Core Integration



Step 6: Defining Personas

This reference design defines five separate personas for the parent and child PR partitions. To define and include the personas in your project:

1. Create four SystemVerilog files, `blinking_led_child.sv`, `blinking_led_child_slow.sv`, `blinking_led_child_empty.sv`, and `blinking_led_slow.sv` in your working directory for the five personas.

   Note: If you create the SystemVerilog files from the Intel Quartus Prime Text Editor, disable the **Add file to current project** option, when saving the files.

Table 2. Reference Design Personas

File Name	Description	Code
`blinking_led_child.sv`	Default persona for the child-level design	`` `timescale 1 ps / 1 ps `` `` `default_nettype none `` `module blinking_led_child (// clock input wire clock, input wire [31:0] counter, // Control signals for the LEDs output wire led_three_on); localparam COUNTER_TAP = 23; reg led_three_on_r; assign led_three_on = led_three_on_r; always_ff @(posedge clock) begin led_three_on_r <= counter[COUNTER_TAP];`

continued...

File Name	Description	Code

		`end`
		
		`endmodule`

`blinking_led_child_slow.sv`	The LED_THREE blinks slower	`` `timescale 1 ps / 1 ps ``
		`` `default_nettype none ``
		
		`module blinking_led_child_slow (`
		
		`// clock`
		`input wire clock,`
		`input wire [31:0] counter,`
		
		`// Control signals for the LEDs`
		`output wire led_three_on`
		`);`
		
		`localparam COUNTER_TAP = 27;`
		`reg led_three_on_r;`
		
		`assign led_three_on = led_three_on_r;`
		
		`always_ff @(posedge clock) begin`
		`led_three_on_r <= counter[COUNTER_TAP];`
		`end`
		
		`endmodule`

`blinking_led_child_empty.sv`	The LED_THREE stays ON	`` `timescale 1 ps / 1 ps ``
		`` `default_nettype none ``
		
		`module blinking_led_child_empty (`
		
		`// clock`
		`input wire clock,`
		`input wire [31:0] counter,`
		
		`// Control signals for the LEDs`
		`output wire led_three_on`
		
		`);`
		
		`// LED is active low`
		`assign led_three_on = 1'b0;`
		
		`endmodule`

`blinking_led_slow.sv`	The LED_TWO blinks slower.	`` `timescale 1 ps / 1 ps ``
		`` `default_nettype none ``
		
		`module blinking_led_slow(`
		
		`// clock`
		`input wire clock,`
		`input wire [31:0] counter,`
		
		`// Control signals for the LEDs`
		`output wire led_two_on,`
		`output wire led_three_on`
		
		`);`
		
		`localparam COUNTER_TAP = 27;`
		
		`reg led_two_on_r;`
		`assign led_two_on = led_two_on_r;`
		
		`// The counter:`
		`always_ff @(posedge clock) begin`
		`led_two_on_r <= counter[COUNTER_TAP];`
		`end`

continued...

File Name	Description	Code

		`blinking_led_child u_blinking_led_child(`
		`.led_three_on (led_three_on),`
		`.counter (counter),`
		`.clock (clock)`
		`);`
		
		`endmodule`

Related Information

Step 3: Creating Design Partitions on page 8

Step 7: Creating Revisions

The PR design flow uses the project revisions feature in the Intel Quartus Prime software. Your initial design is the base revision, where you define the static region boundaries and reconfigurable regions on the FPGA.

From the base revision, you create multiple revisions. These revisions contain the different implementations for the PR regions. However, all PR implementation revisions use the same top-level placement and routing results from the base revision.

To compile a PR design, you must create a PR implementation revision for each persona. In addition, you must assign revision types for each of the revisions. There are the following revision types:

- Partial Reconfiguration - Base
- Partial Reconfiguration - Persona Implementation

Note: The new simplified PR flow introduced in the current version of the Intel Quartus Prime Pro Edition software does not require a separate synthesis and implementation revision for additional PR personas.

The following table lists the revision name and the revision type for each of the revisions:

Table 3. Revision Names and Types

Revision Name	Revision Type
`blinking_led.qsf`	Partial Reconfiguration - Base
`hpr_child_default.qsf`	Partial Reconfiguration - Persona Implementation
`hpr_child_slow.qsf`	Partial Reconfiguration - Persona Implementation
`hpr_child_empty.qsf`	Partial Reconfiguration - Persona Implementation
`hpr_parent_slow_child_default.qsf`	Partial Reconfiguration - Persona Implementation
`hpr_parent_slow_child_slow.qsf`	Partial Reconfiguration - Persona Implementation

Table 4. Parent and Child Persona Revisions

Revision Name	Parent Persona Behavior	Child Persona Behavior
`hpr_child_default.qsf`	Fast blinking	Fast blinking
`hpr_child_slow.qsf`	Fast blinking	Slow blinking

continued...

Revision Name	Parent Persona Behavior	Child Persona Behavior
`hpr_child_empty.qsf`	Fast Blinking	No blinking (always ON)
`hpr_parent_slow_child_default.qsf`	Slow blinking	Fast blinking
`hpr_parent_slow_child_slow.qsf`	Slow blinking	Slow blinking

Setting the Base Revision Type

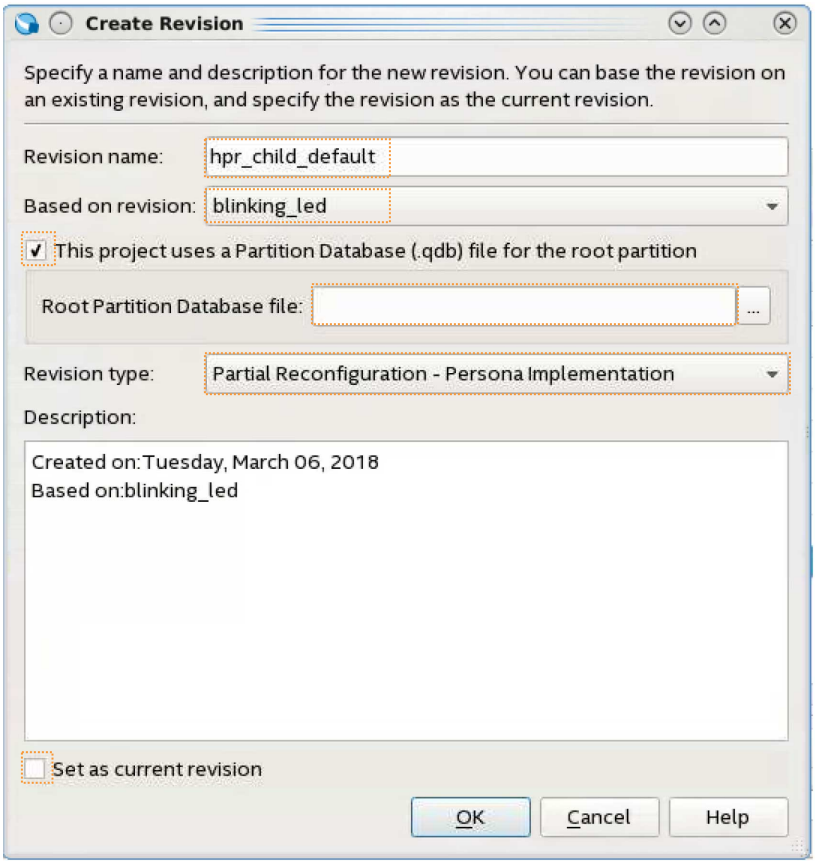
1. Click **Project ➤ Revisions**.
2. In **Revision Name**, select the **blinking_led** revision, and then click **Set Current**.
3. Click **Apply**. The `blinking_led` revision displays as the current revision.
4. To set the **Revision Type** for `blinking_led`, click **Assignments ➤ Settings ➤ General**.
5. For **Revision Type**, select **Partial Reconfiguration - Base**, and then click **OK**.
6. Verify that the `blinking_led.qsf` now contains the following assignment:

```
##blinking_led.qsf
set_global_assignment -name REVISION_TYPE PR_BASE
```

Creating Implementation Revisions

1. To open the **Revisions** dialog box, click **Project ➤ Revisions**.
2. To create a new revision, double-click **<<new revision>>**.
3. In **Revision name**, specify `hpr_child_default` and select **blinking_led** for **Based on revision**.
4. For the **Revision type**, select **Partial Reconfiguration - Persona Implementation**.
5. Enable **This project uses a Partition Database (.qdb) file for the root partition**. You do not need to specify the **Root Partition Database file** at this point. You can input this name at a later stage from the **Design Partitions Window**.

Figure 9. Creating Revisions



6. Similarly, set the **Revision type** for the other revisions:
   - `hpr_child_slow`
   - `hpr_child_empty`
   - `hpr_parent_slow_child_default`
   - `hpr_parent_slow_child_slow`

   Note: Do not specify the above revisions as current revision.
7. Verify that each `.qsf` file now contains the following assignment:

```
set_global_assignment -name REVISION_TYPE PR_IMPL
```

Step 8: Compiling the Base Revision and Exporting the Static Region

Before you begin:

1. Run the PR bitstream ID init script using the following command:

```
quartus_sh -t prpof_id_mif_gen.tcl init
```

This command allows the Intel Quartus Prime software to assign bitstream IDs to child PR regions, for bitstream compatibility check.

2. Ensure the `blinking_led.qsf` contains the following assignments:

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

These assignments allow the assembler to automatically generate the required PR bitstreams.

To compile the base revision and export the static region:

1. To compile the base revision, click **Processing ➤ Start Compilation**. Alternatively, the following command compiles the base revision:

```
quartus_sh --flow compile  blinking_led -c blinking_led
```

2. To regenerate the base `.sof` file with the proper bitstream IDs for the child PR regions, run the PR bitstream ID update script using the following command:

```
quartus_sh -t prpof_id_mif_gen.tcl update
```

3. To export the root partition, click **Project ➤ Export Design Partition**, and then specify the following options for the partition:

Option	Setting
Partition name	**root_partition**
Partition database file	<project>/blinking_led_static.qdb
Include entity-bound SDC files	Enable
Snapshot	**Final**

Figure 10. Exporting the Static Region

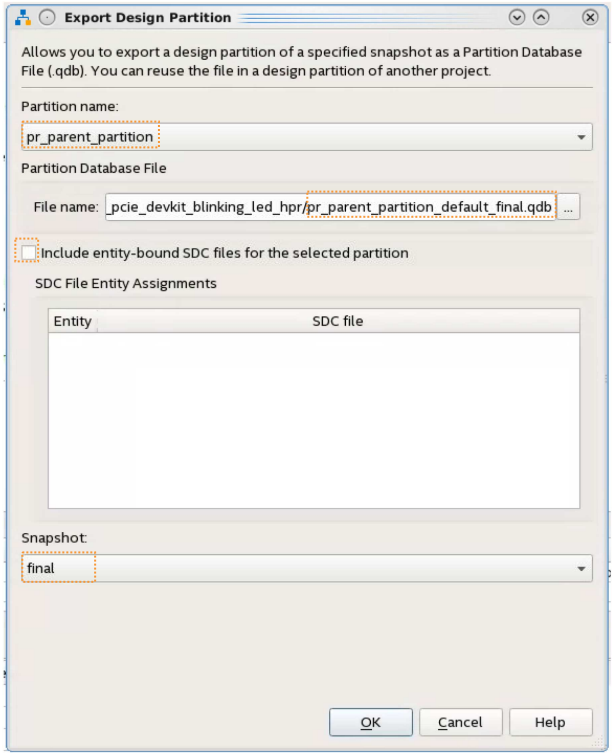
Alternatively, the following command exports the root partition:

```
quartus_cdb -r  blinking_led -c blinking led --export_block \
    root_partition --snapshot final --file blinking_led_static.qdb
```

4. To export the parent PR partition, click **Project ➤ Export Design Partition**, and then specify the following options for the partition:

Option	Setting
Partition name	**pr_parent_partition**
Partition database file	`<project>/ pr_parent_partition_default_final.qdb`
Include entity-bound SDC files	Enable
Snapshot	**Final**

Figure 11. Exporting the Parent PR Region



Alternatively, the following command exports the parent PR region:

```
quartus_cdb -r  blinking_led -c blinking led --export_block \
    root_partition --snapshot final --file
pr_parent_partition_default_final.qdb
```

5. Inspect the bitstream files generated to the `output_files` directory.

Table 5. Generated Bitstream Files

Name	Type	Description
`blinking_led.sof`	Base programming file	Used to program the FPGA with the static logic, along with the default personas for the parent and child PR regions.
`blinking_led.pr_parent_partition.rbf`	PR bitstream file for parent PR partition	Used to program the default persona for the parent PR region.
`blinking_led.pr_parent_partition.pr_partition.rbf`	PR bitstream file for child PR partition	Used to program the default persona for the child PR region.

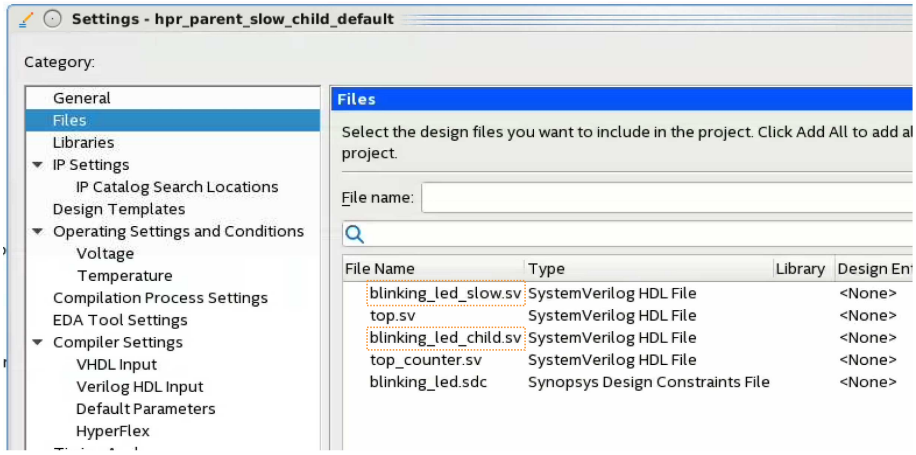
Related Information

- Floorplan the Partial Reconfiguration Design
- Applying Floorplan Constraints Incrementally

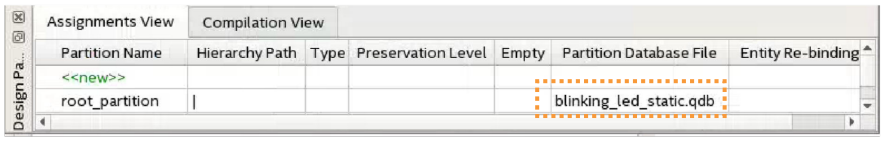
Step 9: Preparing the PR Implementation Revisions for Parent PR Partition

You must prepare the parent and child PR implementation revisions before you can generate the PR bitstream for device programming. This setup includes mapping the new PR logic to the preexisting parent PR partition.

1. To set the current revision, click **Project ➤ Revisions**, select **hpr_parent_slow_child_default** as the **Revision name**, and then click **Set Current**.
2. To verify the correct source for each implementation revision, click **Project ➤ Add/Remove Files in Project**. The `blinking_led_child.sv` file appears in the file list.



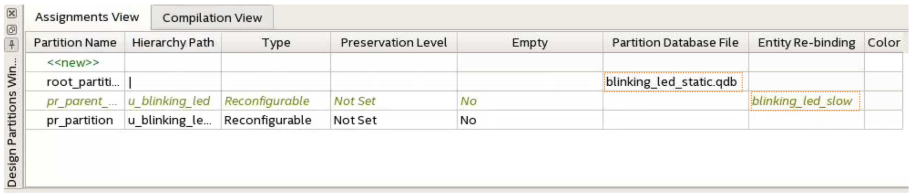
3. To specify the `.qdb` file associated with the static region, click **Assignments ➤ Design Partitions Window**. Double-click the **Partition Database File** cell and navigating to the `blinking_led_static.qdb` file.



Alternatively, the following command assigns this file:

```
set_instance_assignment -name QDB_FILE_PARTITION \
    blinking_led_static.qdb -to |
```

4. In the **Entity** cell, specify the entity name the PR parent partition. For this implementation revision, the entity name is `blinking_led_slow`. `blinking_led_slow` is the name of the entity that you are partially reconfiguring. `u_blinking_led` is the name of the instance that your entity overwrites during PR. Verify that the following line now exists in the `.qsf`:




```
#hpr_parent_slow_child_default.qsf
set_instance_assignment -name ENTITY_REBINDING blinking_led_slow -to
u_blinking_led
```

Note: Because the child PR logic is already defined by the parent PR partition, whose entity name is rebound, do not use an entity rebinding assignment for the child PR partition.

5. Before compiling the implementation revision, ensure the corresponding `.qsf` file contains the following assignments:

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

These assignments allow the assembler to automatically generate the required PR bitstreams.

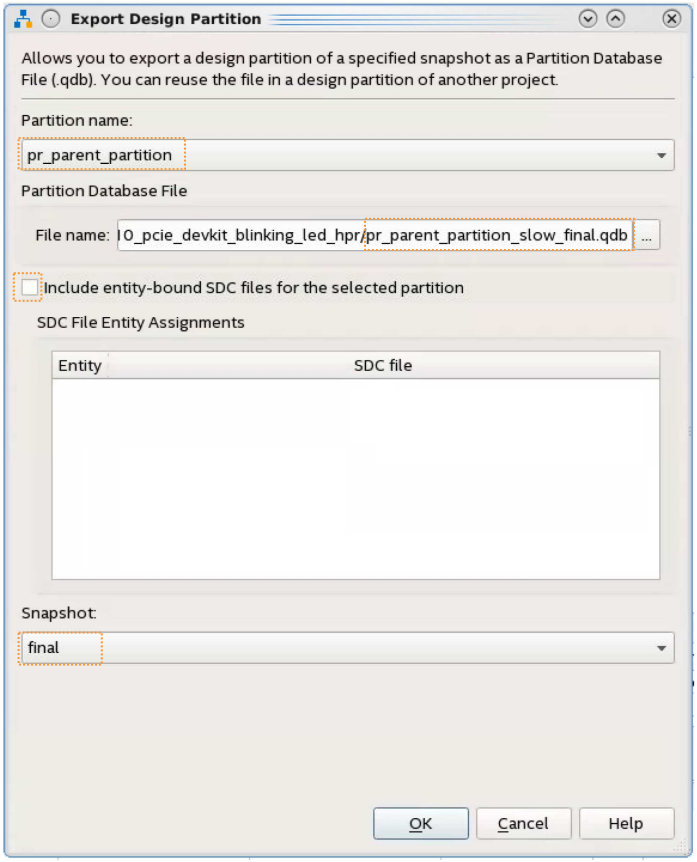
6. To compile the design, click **Processing ➤ Start Compilation**. Alternatively, the following command compiles this project:

```
quartus_sh --flow compile blinking_led –c hpr_parent_slow_child_default
```

7. To export this new parent PR partition as a finalized `.qdb`, click **Project ➤ Export Design Partition**, and then specify the following options for the partition:

Option	Setting
Partition name	**pr_parent_partition**
Partition database file	`<project>/pr_parent_partition_slow_final.qdb`
Include entity-bound SDC files	Enable
Snapshot	**Final**

Figure 12. Exporting the Modified Parent PR Region



Alternatively, the following command exports the parent PR region:

```
quartus_cdb -r  blinking_led -c blinking led --export_block \
    root_partition --snapshot final --file
pr_parent_partition_slow_final.qdb
```

8. Inspect the bitstream files generated to the `output_files` directory.

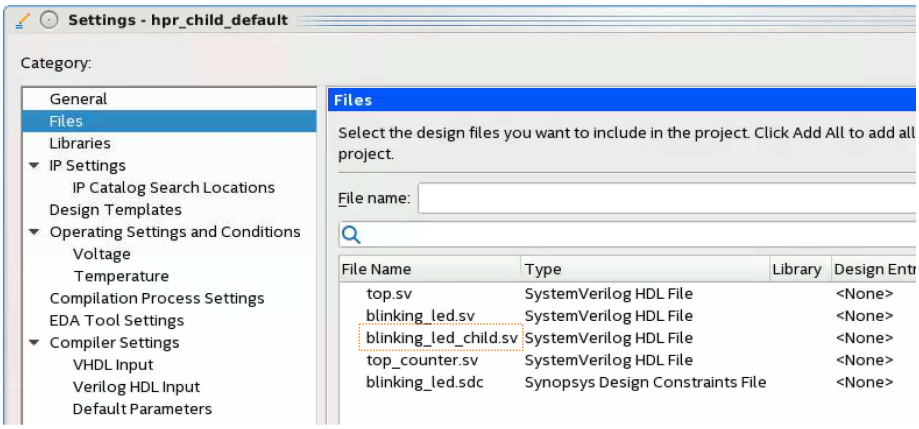
Table 6. Generated Bitstream Files

Name	Type	Description
`hpr_parent_slow_child_default.pr_parent_partition.rbf`	PR bitstream file for parent PR partition	Used to program the default persona for the parent PR region. Causes the `led_two_on` to blink at a lower rate.
`hpr_parent_slow_child_default.pr_parent_partition.pr_partition.rbf`	PR bitstream file for child PR partition	Used to program the default persona for the child PR region. Causes the `led_three_on` to blink at the default rate.

Step 10: Preparing the PR Implementation Revisions for Child PR Partitions

This setup includes adding the static region `.qdb` file as the source file for each implementation revision. In addition, you must import the parent PR partition `.qdb` file and specify the corresponding entity of the PR region.

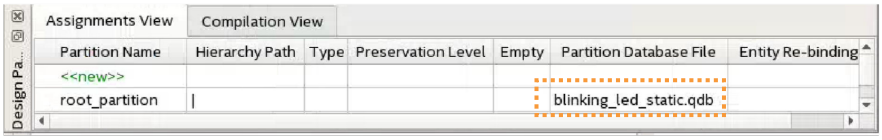
1. To set the current revision, click **Project ➤ Revisions**, select **hpr_child_default** as the **Revision name**, and then click **Set Current**.
2. To verify the correct source for each implementation revision, click **Project ➤ Add/Remove Files in Project**. The `blinking_led_child.sv` file appears in the file list.



3. Repeat steps 1 through 2 to verify the other implementation revision source files:

Implementation Revision Name	Child Persona Source File
`hpr_child_default`	`blinking_led_child.sv`
`hpr_child_slow`	`blinking_led_child_slow.sv`
`hpr_child_empty`	`blinking_led_child_empty.sv`
`hpr_parent_slow_child_slow`	`blinking_led_child_slow.sv`

4. To verify the `.qdb` file associated with the root partition, click **Assignments ➤ Design Partitions Window**. Specify the `.qdb` file associated with the static region by double-clicking the **Partition Database File** cell and navigating to the `blinking_led_static.qdb` file.



Alternatively, the following command assigns this file:

```
set_instance_assignment -name QDB_FILE_PARTITION \
                        blinking_led_static.qdb -to |
```

5. To specify the parent PR partition `.qdb` file, click **Assignments ➤ Design Partitions Window**. Double-click the **Partition Database File** for the `parent_pr_partition` and specify the respective `.qdb` file.

Implementation Revision Name	Parent Persona .qdb File
hpr_child_default	pr_parent_partition_default_final.qdb
hpr_child_slow	pr_parent_partition_default_final.qdb
hpr_child_empty	pr_parent_partition_default_final.qdb
hpr_parent_slow_child_slow	pr_parent_partition_slow_final.qdb

verify the following line exists in the `.qsf`:

```
# To use the default parent PR persona:
set_instance_assignment -name QDB_FILE_PARTITION \
      pr_parent_partition_default_final.qdb -to u_blinking_led

# To use the slow parent PR persona:
set_instance_assignment -name QDB_FILE_PARTITION \
      pr_parent_partition_slow_final.qdb -to u_blinking_led
```

6. In the **Entity Re-binding** cell, specify the entity name of the child PR partition. For the default persona, the entity name is `blinking_led`. For this implementation revision, `blinking_led_child` is the name of the entity that you are partially reconfiguring. `u_blinking_led|u_blinking_led_child` is the name of the instance that your entity overwrites during PR. Verify that the following line now exists in the `.qsf`:


```
#hpr_child_default.qsf
set_instance_assignment -name ENTITY_REBINDING \
      blinking_led_child -to u_blinking_led|u_blinking_led_child

#hpr_child_slow.qsf and hpr_parent_slow_child_slow.qsf
set_instance_assignment -name ENTITY_REBINDING \
      blinking_led_child_slow -to u_blinking_led|u_blinking_led_child

#hpr_child_empty.qsf
set_instance_assignment -name ENTITY_REBINDING \
      blinking_led_child_empty -to u_blinking_led|u_blinking_led_child
```

7. Before compiling the implementation revision, ensure the corresponding `.qsf` file contains the following assignments:

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

These assignments allow the assembler to automatically generate the required PR bitstreams.

8. To compile the design, click **Processing ➤ Start Compilation**. Alternatively, the following command compiles this project:

```
quartus_sh --flow compile blinking_led –c hpr_child_default
```

9. Repeat the above steps to prepare `hpr_child_slow`, `hpr_child_empty`, and `hpr_parent_slow_child_slow` revisions.

Note: You can specify any Fitter specific settings that you want to apply during the PR implementation compilation. Fitter specific settings impact only the fit of the persona, without affecting the imported static region.

10. Inspect the bitstream files generated to the `output_files` directory.

    Verify that the `output_files` directory contains the following generated `.rbf` files after compiling all the implementation revisions:

    - `hpr_child_default.pr_parent_partition.rbf`
    - `hpr_child_slow.pr_parent_partition.rbf`
    - `hpr_child_empty.pr_parent_partition.rbf`
    - `hpr_parent_slow_child_slow.pr_parent_partition.rbf`
    - `hpr_child_default.pr_parent_partition.pr_partition.rbf`
    - `hpr_child_slow.pr_parent_partition.pr_partition.rbf`
    - `hpr_child_empty.pr_parent_partition.pr_partition.rbf`
    - `hpr_parent_slow_child_slow.pr_parent_partition.pr_partition.rbf`

Step 11: Programming the Board

Before you begin:

1. Connect the power supply to the Arria 10 GX FPGA development board.
2. Connect the USB Blaster cable between your PC USB port and the USB Blaster port on the development board.

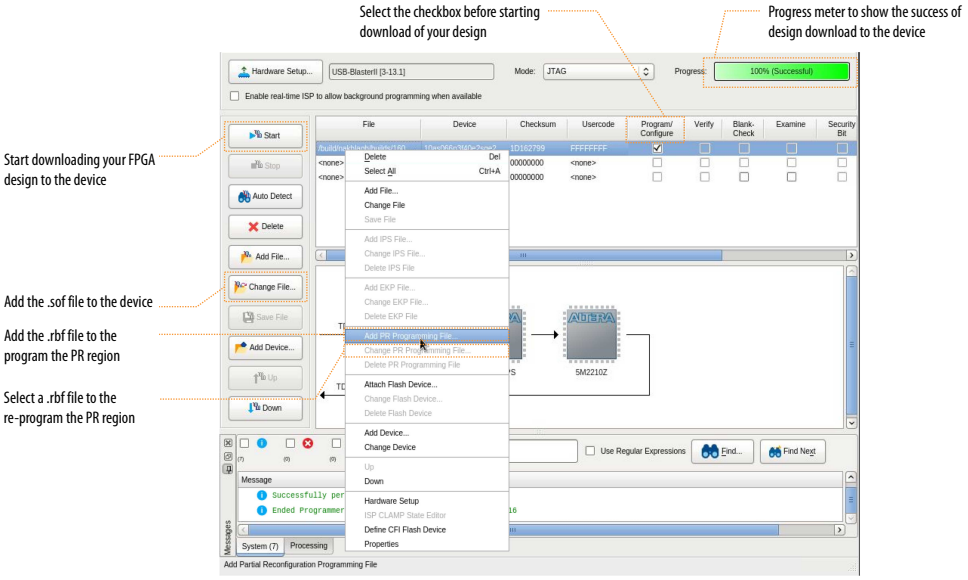
Note: This tutorial utilizes the Arria 10 GX FPGA development board on the bench, outside of the PCIe slot in your host machine.

To run the design on the Arria 10 GX FPGA development board:

1. Open the Intel Quartus Prime software and click **Tools ➤ Programmer**.
2. In the Programmer, click **Hardware Setup** and select **USB-Blaster**.
3. Click **Auto Detect** and select the device, **10AX115S2**.
4. Click **OK**. The Intel Quartus Prime software detects and updates the Programmer with the three FPGA chips on the board.
5. Select the 10AX115S2 device, click **Change File** and load the `blinking_led.sof` file.
6. Enable **Program/Configure** for `blinking_led.sof` file.
7. Click **Start** and wait for the progress bar to reach 100%.
8. Observe the LEDs on the board blinking at the same frequency as the original flat design.
9. To program only the child PR region, right-click the `blinking_led.sof` file in the Programmer and click **Add PR Programming File**.
10. Select the `hpr_child_slow.pr_parent_partition.pr_partition.rbf` file.
11. Disable **Program/Configure** for `blinking_led.sof` file.

12. Enable **Program/Configure** for `hpr_child_slow.pr_parent_partition.pr_partition.rbf` file and click **Start**. On the board, observe `LED[0]` and `LED[1]` continuing to blink. When the progress bar reaches 100%, `LED[2]` blinks at the same rate, and `LED[3]` blinks slower.
13. To program both the parent and child PR region, right-click the `.rbf` file in the Programmer and click **Change PR Programing File**.
14. Select the `hpr_child_slow.pr_parent_partition.rbf` file.
15. Click **Start**. On the board, observe that `LED[0]` and `LED[1]` continuing to blink. When the progress bar reaches 100%, both `LED[2]` and `LED[3]` blink slower.
16. Repeat the above steps to dynamically re-program just the child PR region, or both the parent and child PR regions simultaneously.

Figure 13. Programming the Arria 10 GX FPGA Development Board



Programming the Child PR Region

It is very important that you program the correct child persona to match the parent persona. Running the `prpof_id_mif_gen.tcl` script before and after the base revision compile allows the tool to check for incompatible bitstreams for Arria 10 devices and outputs a PR_ERROR message, indicating the usage of incorrect bitstream. If you do not run this script, you can run into any one of the following types of errors:

- Successful PR programming, but corrupted FPGA functionality
- Unsuccessful PR programming, and corrupted FPGA functionality

If you wish to reprogram a child PR region on the FPGA, you must ensure that the child PR `.rbf` is generated from an implementation revision compile whose parent PR persona matches the persona currently on the FPGA. For example, when you program

the base `blinking_led.sof` onto the FPGA, the parent PR persona is `default`. The child PR persona is `default` as well. To change the child PR persona to `slow` persona, you have the choice of using the following bitstreams:

1. `hpr_child_slow.pr_parent_partition.pr_partition.rbf`
2. `hpr_parent_slow_child_slow.pr_parent_partition.pr_partition.rbf`

In this case, you must choose the former, as this file is generated by an implementation revision that has the `default` parent persona. Choosing the latter (`hpr_parent_slow_child_slow.pr_parent_partition.pr_partition.rbf`) results in unsuccessful PR programming, corrupted FPGA functionality, or both.

Troubleshooting PR Programming Errors

Ensuring proper setup of the Intel Quartus Prime Programmer and connected hardware helps to avoid any errors during PR programming.

If you face any PR programming errors, refer to *Troubleshooting PR Programming Errors* in the *Partial Reconfiguration User Guide* for step-by-step troubleshooting tips.

Related Information

Troubleshooting PR Programming Errors

Modifying an Existing Persona

You can change an existing persona, even after fully compiling the base revision.

For example, to cause the `blinking_led_child_slow` persona to blink even slower:

1. In the `blinking_led_child_slow.sv` file, modify the `COUNTER_TAP` parameter from 27 to 28.
2. Recompile any implementation revision that uses this source file, such as `hpr_child_slow` or `hpr_parent_slow_child_slow`.
3. Regenerate the PR bitstreams from the `.pmsf` files.
4. Follow the steps in Step 11: Programming the Board on page 27 to program the resulting RBF file into the FPGA.

Adding a New Persona to the Design

After fully compiling your base revisions, you can still add new personas and individually compile these personas.

For example, to define a new persona that causes `led_two` (parent) to blink at a slower rate, while keeping `led_three` (child) on:

1. Create an implementation revision, `hpr_parent_slow_child_empty`, by following the steps in Creating Implementation Revisions on page 17.
2. Compile the revision by clicking **Processing ➤ Start Compilation**.

For complete information on hierarchical partial reconfiguration for Arria 10 devices, refer to *Creating a Partial Reconfiguration Design* in Volume 1 of the *Intel Quartus Prime Pro Edition Handbook*.

Related Information

- Creating a Partial Reconfiguration Design
- Partial Reconfiguration Online Training

Document Revision History for AN 806: Hierarchical Partial Reconfiguration Tutorial for Intel Arria 10 GX FPGA Development Board

Document Version	Intel Quartus Prime Version	Changes
2018.05.07	18.0.0	• Compilation flow change • Other minor text edits
2017.11.06	17.1.0	• Updated the *Reference Design Requirements* section with software version • Updated the *Flat Reference Design without PR Partitioning* figure with design block changes • Updated the *Reference Design Files* table with information on the `Top_counter.sv` module • Updated the *Partial Reconfiguration IP Core Integration* figure with design block changes • Updated the figures - *Design Partitions Window* and *Logic Lock Regions Window* to reflect the new GUI • File name changes • Text edits
2017.05.08	17.0.0	Initial release of the document