



# **AN 805: Hierarchical Partial Reconfiguration of a Design on Intel<sup>®</sup> Arria<sup>®</sup> 10 SoC Development Board**

***AN-805  
2017.05.08***



**Subscribe**



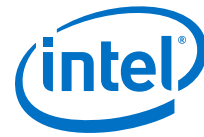
**Send Feedback**



## Contents

---

<b>Hierarchical Partial Reconfiguration of a Design.....</b>	<b>3</b>
Reference Design Requirements.....	3
Reference Design Overview.....	4
Reference Design Files.....	4
Reference Design Walkthrough.....	6
Step 1: Getting Started.....	6
Step 2: Creating a Parent Level Sub-module.....	6
Step 3: Creating Design Partitions.....	8
Step 4: Allocating Placement and Routing Region for PR Partitions.....	9
Step 5: Adding the Partial Reconfiguration Controller IP Core.....	11
Step 6: Defining Personas.....	14
Step 7: Creating Revisions .....	17
Step 8: Generating the Hierarchical Partial Reconfiguration Flow Script.....	22
Step 9: Running the Hierarchical Partial Reconfiguration Flow Script.....	23
Step 10: Programming the Board.....	24
Modifying an Existing Persona.....	25
Adding a New Persona to the Design.....	26
Document Revision History.....	27



## Hierarchical Partial Reconfiguration of a Design

---

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

Hierarchical partial reconfiguration (HPR) is a special type of 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 child partition does not impact operation in the parent or static regions. 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

*Note:*

- Implementation of this reference design requires basic familiarity with the Intel® Quartus® Prime FPGA implementation flow and knowledge of the primary Quartus Prime project files.

### Related Links

- [Arria 10 SoC Development Kit User Guide](#)
- [Partial Reconfiguration Concepts](#)
- [Partial Reconfiguration Design Flow](#)
- [Partial Reconfiguration Design Recommendations](#)
- [Partial Reconfiguration Design Considerations](#)

## Reference Design Requirements

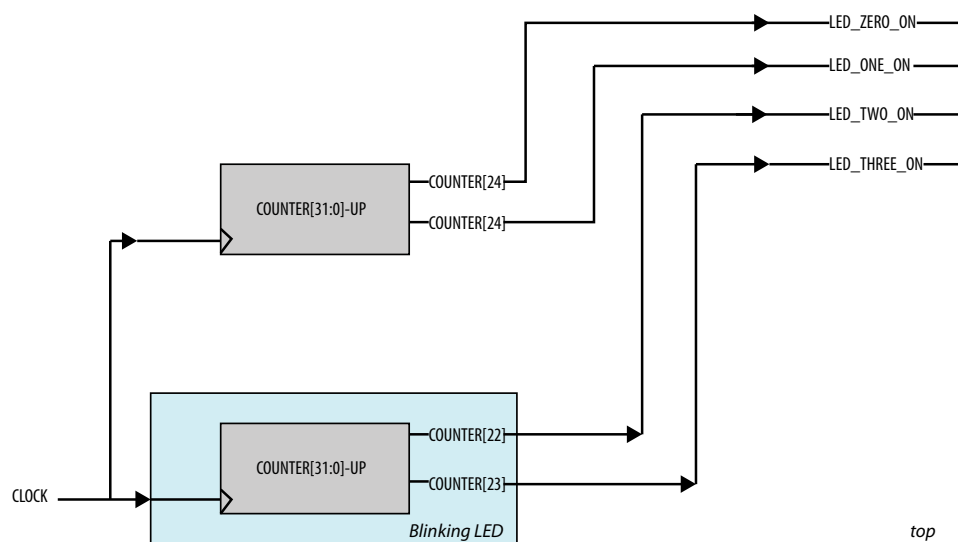
This reference design requires the following:

- Quartus Prime Pro Edition software version 17.0 for the design implementation.
- Arria 10 SoC development kit for the FPGA implementation.

## Reference Design Overview

This reference design consists of two 32-bit counters. At the board level, the design connects the clock to a 50MHz source, and connects the output to 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/01org/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_soc_devkit_blinking_led_hpr` sub-folder to access the reference design.

The `flat` folder consists of the following files:

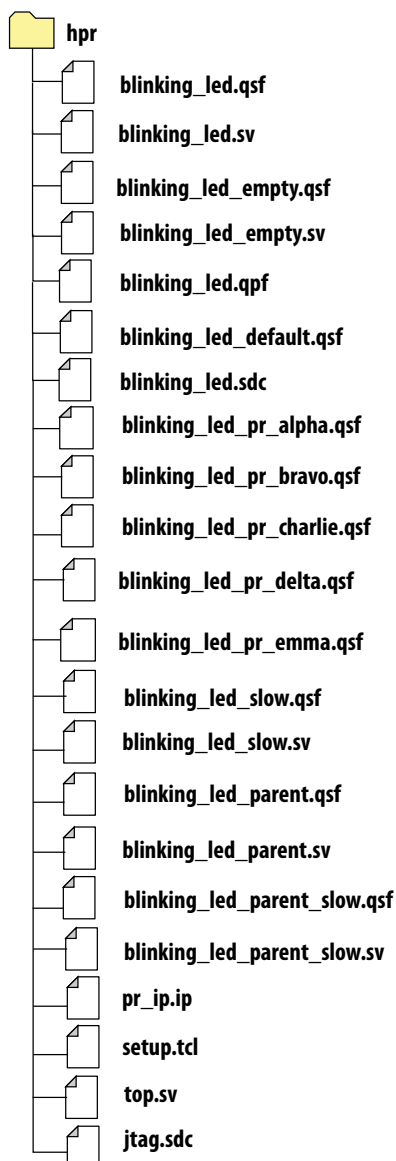
**Table 1. Reference Design Files**

File Name	Description
<code>top.sv</code>	Top-level file containing the flat implementation of the design. This module instantiates the <code>blinking_led</code> sub-partition. The counter at the top-level controls <code>LED[0]</code> and <code>LED[1]</code> .
<code>blinking_led.sdc</code>	Defines the timing constraints for the project.
<i>continued...</i>	

File Name	Description
blinking_led.sv	SystemVerilog file that causes the LEDs to blink using a 32-bit counter. The counter controls <code>LED[2]</code> and <code>LED[3]</code> . You convert this module into a child PR partition, later in this tutorial.
blinking_led.qpf	Quartus Prime project file containing the base revision information.
blinking_led.qsf	Quartus Prime settings file containing the assignments and settings for the project.

**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 Quartus Prime Pro Edition software for the Arria 10 SoC development board:

- [Step 1: Getting Started](#) on page 6
- [Step 2: Creating a Parent Level Sub-module](#) on page 6
- [Step 3: Creating Design Partitions](#) on page 8
- [Step 4: Allocating Placement and Routing Region for PR Partitions](#) on page 9
- [Step 5: Adding the Partial Reconfiguration Controller IP Core](#) on page 11
- [Step 6: Defining Personas](#) on page 14
- [Step 7: Creating Revisions](#) on page 17
- [Step 8: Generating the Hierarchical Partial Reconfiguration Flow Script](#) on page 22
- [Step 9: Running the Hierarchical Partial Reconfiguration Flow Script](#) on page 23
- [Step 10: Programming the Board](#) on page 24

### 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_soc_devkit_blinking_led_hpr`.
2. Copy the downloaded `tutorials/a10_soc_devkit_blinking_led_hpr/flat` sub-folder to the directory, `a10_soc_devkit_blinking_led_hpr`.
3. In the 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 Parent Level Sub-module

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

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

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

module blinking_led_parent(
    // Control signals for the LEDs
    led_two_on,
    led_three_on,

    // clock
    clock
);
```



```
// assuming single bit control signal to turn LED 'on'
output wire    led_two_on;
output wire    led_three_on;

// clock
input wire     clock;

reg [31:0]     count;

localparam COUNTER_TAP = 23;

// The counter:
always_ff @(posedge clock)
begin
    count <= count + 1;
end

assign led_two_on = count[COUNTER_TAP];

blinking_led u_blinking_led(
    .led_three_on    (led_three_on),
    .clock           (clock)
);

endmodule
```

2. Modify the `blinking_led.sv` file to remove all instances of `led_two_on`, so that the child sub-module contains only one LED signal. After modifications, your `blinking_led.sv` file must appear as follows:

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

module blinking_led (
    // Control signals for the LEDs
    led_three_on,

    // clock
    clock
);

// assuming single bit control signal to turn LED 'on'
output wire    led_three_on;

// clock
input wire     clock;

reg [31:0]     count;

localparam COUNTER_TAP = 23;

// The counter:
always_ff @(posedge clock)
begin
    count <= count + 1;
end

assign led_three_on = count[COUNTER_TAP];

endmodule
```

3. Replace the following lines of code in `top.sv`:

```
blinking_led u_blinking_led
(
    .led_two_on    (led_two_on_w),          // used in flat implementation
    // .led_two_on    (pr_led_two_on),      // used in PR implementation
```

```
.led_three_on (led_three_on_w),      // used in flat implementation
//      .led_three_on (pr_led_three_on), // used in PR implementation

.clock      (clock)
);
```

with the following lines of code, so that the top-level design instantiates the parent level sub-module:

```
blinking_led_parent u_blinking_led_parent
(
    .led_two_on      (led_two_on_w),      //used in flat implementation
    //      .led_two_on      (pr_led_two_on),      //used in PR implementation

    .led_three_on    (led_three_on_w),    //used in flat implementation
    //      .led_three_on    (pr_led_three_on),    //used in PR implementation

    .clock           (clock)
);
```

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

### Step 3: Creating Design Partitions

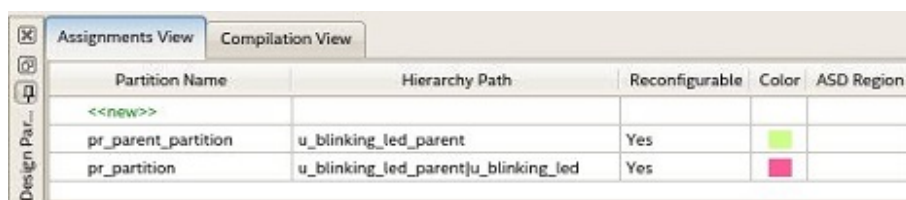
You must create design partitions for each PR region that you want to partially reconfigure. You can create any number of independent partitions or PR regions in your design. This tutorial creates two design partitions for the `u_blinking_led_parent` and `u_blinking_led` instances.

To create design partitions for hierarchical partial reconfiguration:

1. Right-click the `u_blinking_led_parent` instance in the **Project Navigator** and click **Design Partition ► Set as Reconfigurable Design Partition**.

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

**Figure 3. Design Partitions Window**



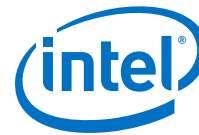
Partition Name	Hierarchy Path	Reconfigurable	Color	ASD Region
<<new>>				
pr_parent_partition	u_blinking_led_parent	Yes	Green	
pr_partition	u_blinking_led_parent u_blinking_led	Yes	Pink	

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

*Note:* When you create a partition, the 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.

3. Repeat steps 1 and 2 to assign reconfigurable design partitions to the `u_blinking_led` instance. Rename this partition to `pr_partition`.





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_parent |  
u_blinking_led  
set_instance_assignment -name PARTIAL_RECONFIGURATION_PARTITION ON -to  
u_blinking_led_parent|u_blinking_led  
  
set_instance_assignment -name PARTITION pr_parent_partition -to  
u_blinking_led_parent  
set_instance_assignment -name PARTIAL_RECONFIGURATION_PARTITION ON -to  
u_blinking_led_parent
```

### Related Links

[Create Design Partitions for Partial Reconfiguration](#)

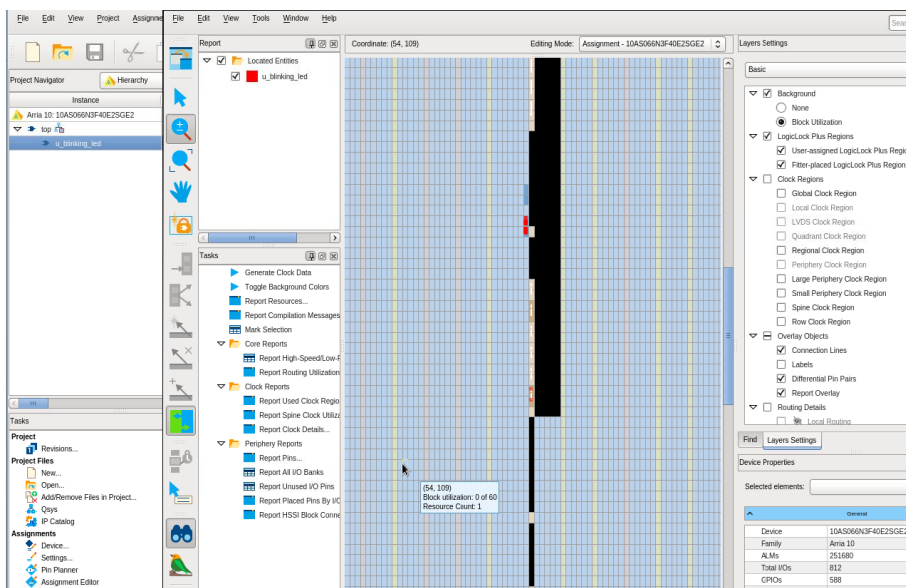
## Step 4: Allocating Placement and Routing Region for PR Partitions

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` instance in the **Project Navigator** and click **LogicLock® Plus Region ► Create New LogicLock Plus Region**. The region appears on the LogicLock Plus Regions Window.
2. Your placement region must enclose the `blinking_led` logic. Select the placement region by locating the node in Chip Planner. Right-click the `u_blinking_led` region name in the **Project Navigator** and click **Locate Node ► Locate in Chip Planner**.

The `u_blinking_led` region is color-coded.

**Figure 4. Chip Planner Node Location for `blinking_led`**



3. In the LogicLock Plus 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 Quartus Prime software automatically calculates the (X2 Y2) co-ordinates (top-right) for the placement region, based on the height and width you specify.

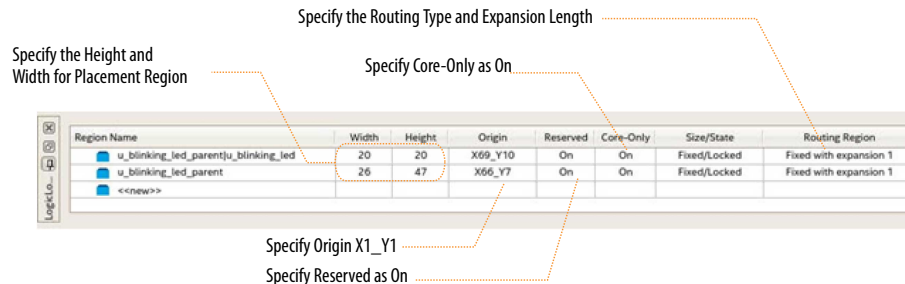
*Note:* This tutorial uses the (X1 Y1) co-ordinates - (69 10), and a height and width of 20 for the placement region. Define any value for the placement region, as long as the region covers the `blinking_led` logic.

4. Enable the **Reserved** and **Core-Only** options.
5. Double-click the **Routing Region** option. The **LogicLock Plus 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 routing engine when the engine routes different personas.

7. Repeat steps 1 -6 for the `u_blinking_led_parent` 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_parent` instance.

Figure 5. LogicLock Plus 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_parent|u_blinking_led
set_instance_assignment -name RESERVE_PLACE_REGION ON -to u_blinking_led_parent|
u_blinking_led
set_instance_assignment -name CORE_ONLY_PLACE_REGION ON -to u_blinking_led_parent|
u_blinking_led
set_instance_assignment -name ROUTE_REGION "68 9 89 30" -to u_blinking_led_parent|
u_blinking_led

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

### Related Links

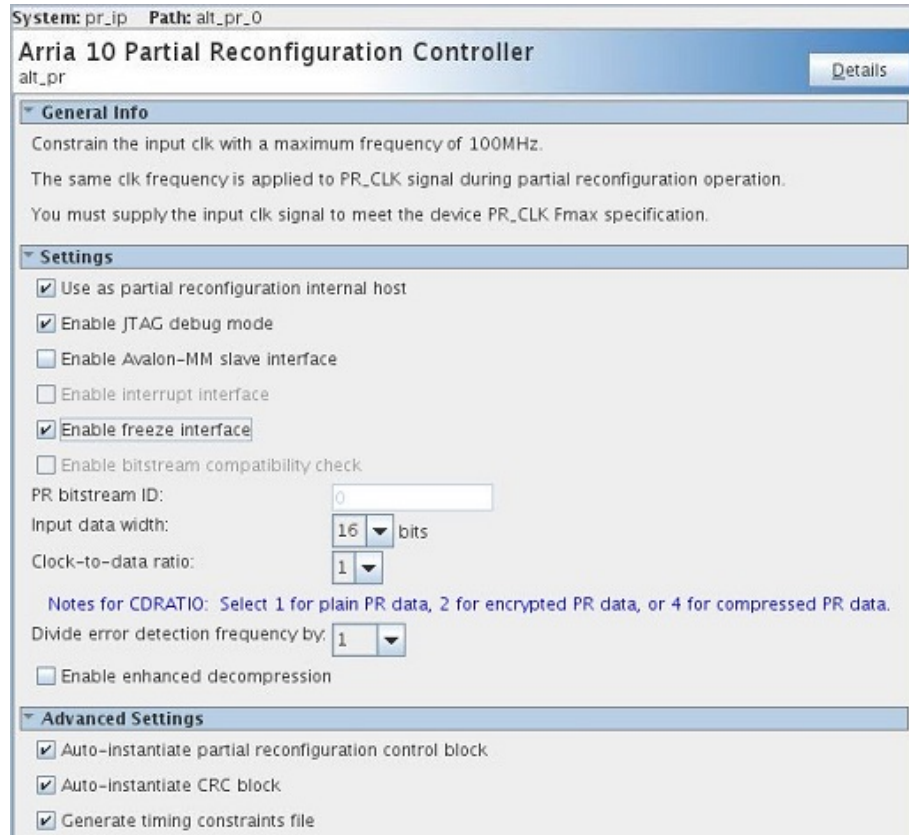
- [Floorplan the Partial Reconfiguration Design](#)
- [Incrementally Implementing Partial Reconfiguration](#)

## Step 5: Adding the Partial Reconfiguration Controller IP Core

Use the Partial Reconfiguration Controller IP core to reconfigure the PR partition. This IP core uses JTAG to reconfigure the PR partition. To add the Partial Reconfiguration Controller IP core to your 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, and **Enable Avalon-MM slave interface** option is turned off.

**Figure 6. Arria 10 Partial Reconfiguration Controller IP Core Parameters**



System: pr\_ip Path: alt\_pr\_0

### Arria 10 Partial Reconfiguration Controller

alt\_pr Details

**General Info**

Constrain the input clk with a maximum frequency of 100MHz.  
 The same clk frequency is applied to PR\_CLK signal during partial reconfiguration operation.  
 You must supply the input clk signal to meet the device PR\_CLK Fmax specification.

**Settings**

☒ Use as partial reconfiguration internal host  
☒ Enable JTAG debug mode  
☐ Enable Avalon-MM slave interface  
☐ Enable interrupt interface  
☒ Enable freeze interface  
☐ Enable bitstream compatibility check

PR bitstream ID:   
 Input data width:  bits  
 Clock-to-data ratio:   
 Notes for CDRATIO: Select 1 for plain PR data, 2 for encrypted PR data, or 4 for compressed PR data.  
 Divide error detection frequency by:   
☐ Enable enhanced decompression

**Advanced Settings**

☒ Auto-instantiate partial reconfiguration control block  
☒ Auto-instantiate CRC block  
☒ Generate timing constraints file

4. Click **Finish**, and exit the parameter editor without generating the system. 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:

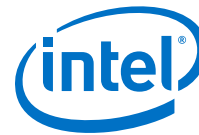
```
set_global_assignment -name IP_FILE pr_ip.ip
```

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 Quartus Prime project revision. For more information, refer to *Timing Constraints* section in the *Partial Reconfiguration IP Core User Guide*.

### Related Links

- [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     ()
);
```

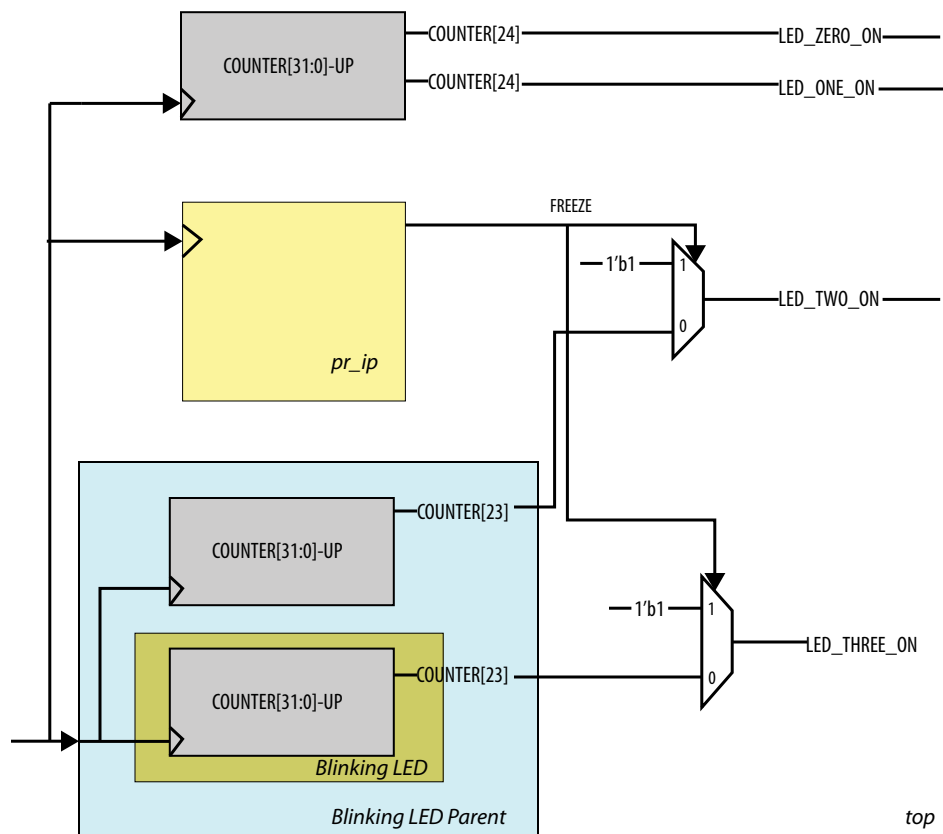
2. To force the output ports to logic 1 during reconfiguration, use the freeze control signal output from `PR_IP`. Uncomment the following lines of code:

```
assign led_two_on_w    = freeze ? 1'b1 : 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_parent`), update the `top.sv` file with the following block of code:

```
blinking_led_parent u_blinking_led_parent
(
    .led_two_on    (pr_led_two_on),
    .led_three_on  (pr_led_three_on),
    .clock         (clock)
);
```

**Figure 7. 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 five SystemVerilog files, `blinking_led.sv`, `blinking_led_slow.sv`, `blinking_led_empty.sv`, `blinking_led_parent.sv`, and `blinking_led_parent_slow.sv` in your working directory for the five personas.

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

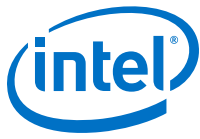
**Table 2. Reference Design Personas**

File Name	Description	Code
<code>blinking_led.sv</code>	Default persona for the child-level design	<pre> `timescale 1 ps / 1 ps `default_nettype none  module blinking_led (     // Control signals for the LEDs     led_three_on,      // clock     clock </pre>
<i>continued...</i>		



File Name	Description	Code
		<pre> );  // assuming single bit control signal to turn LED 'on' output wire led_three_on;  // clock input wire clock;  // the 32-bit counter reg [31:0] count;  localparam COUNTER_TAP = 23;  // The counter: always_ff @(posedge clock) begin     count &lt;= count + 1; end  assign led_three_on = count[COUNTER_TAP];  endmodule </pre>
blinking_led_slow.v	The LED_THREE blinks slower	<pre> `timescale 1 ps / 1 ps `default_nettype none  module blinking_led_slow (      // Control signals for the LEDs     led_three_on,      // clock     clock );  // assuming single bit control signal to turn LED 'on' output wire led_three_on;  // clock input wire clock;  // the 32-bit counter reg [31:0] count;  localparam COUNTER_TAP = 27;  // The counter: always_ff @(posedge clock) begin     count &lt;= count + 1; end  assign led_three_on = count[COUNTER_TAP];  endmodule </pre>
blinking_led_empty.v	The LED_THREE stays ON	<pre> `timescale 1 ps / 1 ps `default_nettype none  module blinking_led_empty (      // Control signals for the LEDs     led_three_on,      // clock     clock );  // Control signal to turn LED 'on' output wire led_three_on;  // clock input wire clock;  // LED is active low </pre>

continued...



File Name	Description	Code
		<pre> assign led_three_on = 1'b0; endmodule </pre>
blinking_led_parent.sv	Default persona for the parent-level design	<pre> `timescale 1 ps / 1 ps `default_nettype none  module blinking_led_parent(     // Control signals for the LEDs     led_two_on,     led_three_on,      // clock     clock );  // assuming single bit control signal to turn LED 'on' output wire led_two_on; output wire led_three_on;  // clock input wire clock;  reg [31:0] count;  localparam COUNTER_TAP = 23;  // The counter: always_ff @(posedge clock) begin     count &lt;= count + 1; end  assign led_two_on = count[COUNTER_TAP];  blinking_led u_blinking_led(     .led_three_on (led_three_on),     .clock (clock) );  endmodule </pre>
blinking_led_parent_slow.sv	The LED_TWO blinks slower	<pre> `timescale 1 ps / 1 ps `default_nettype none  module blinking_led_parent_slow(     // Control signals for the LEDs     led_two_on,     led_three_on,      // clock     clock );  // assuming single bit control signal to turn LED 'on' output wire led_two_on; output wire led_three_on;  // clock input wire clock;  reg [31:0] count;  localparam COUNTER_TAP = 26;  // The counter: always_ff @(posedge clock) begin     count &lt;= count + 1; end  assign led_two_on = count[COUNTER_TAP];  blinking_led u_blinking_led(     .led_three_on (led_three_on),     .clock (clock) );  endmodule </pre>





### Related Links

[Step 3: Creating Design Partitions](#) on page 8

You must create design partitions for each PR region that you want to partially reconfigure.

## Step 7: Creating Revisions

The PR design flow uses the project revisions feature in the 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 and synthesis revision for each persona. In this reference design, in addition to the base revision (`blinking_led`), the three child-level personas and the two parent-level personas contain five separate synthesis revisions, and five separate implementation revisions:

**Table 3. Revisions for the Two Parent Personas and Three Child Personas**

Synthesis Revision	Implementation Revision
<code>blinking_led_parent, blinking_led_default</code>	<code>blinking_led_pr_alpha</code>
<code>blinking_led_parent, blinking_led_slow</code>	<code>blinking_led_pr_bravo</code>
<code>blinking_led_parent, blinking_led_empty</code>	<code>blinking_led_pr_charlie</code>
<code>blinking_led_parent_slow, blinking_led_slow</code>	<code>blinking_led_pr_delta</code>
<code>blinking_led_parent_slow, blinking_led_empty</code>	<code>blinking_led_pr_emma</code>

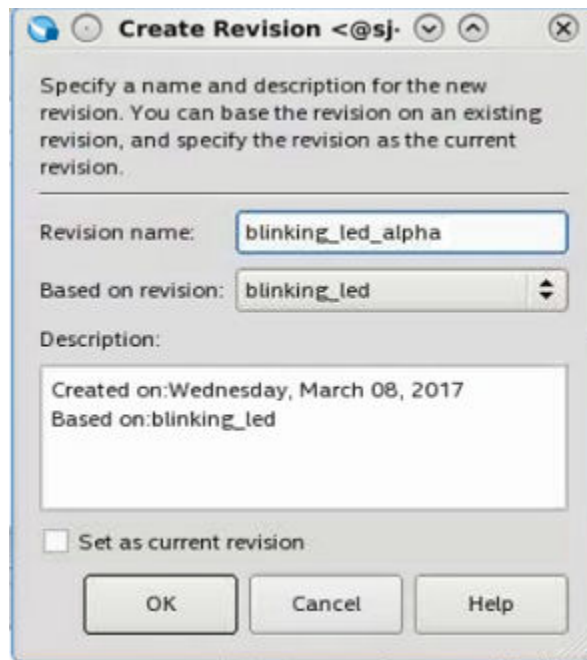
### Creating Implementation Revisions

To create the PR implementation revisions:

1. To open the **Revisions** dialog box, click **Project ► Revisions**.
2. To create a new revision, double-click **<<new revision>>**.
3. Specify the **Revision name** as `blinking_led_pr_alpha` and select `blinking_led` for **Based on Revision**.
4. Disable the **Set as current revision** option and click **OK**.
5. Similarly, create `blinking_led_pr_bravo`, `blinking_led_pr_charlie`, `blinking_led_pr_delta`, and `blinking_led_pr_emma` revisions, based on the `blinking_led` revision.

*Note:* Do not set the above revisions as current revision.

Figure 8. Creating Revisions



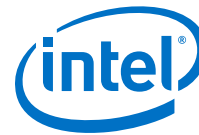
### Creating Synthesis-Only Revisions

To create synthesis-only revisions for the personas, you must assign the top-level entity and corresponding SystemVerilog file for each of the personas:

1. In the Quartus Prime software, click **Project > Revisions**.
2. Create `blinking_led_default` revision based on `blinking_led` revision. Do not set this revision as current revision.
3. Modify `blinking_led_default.qsf` file to include the following assignments:

```
set_global_assignment -name TOP_LEVEL_ENTITY blinking_led
set_global_assignment -name SYSTEMVERILOG_FILE blinking_led.sv
```

4. Similarly, create `blinking_led_slow`, `blinking_led_empty`, `blinking_led_parent`, and `blinking_led_parent_slow` revisions based on `blinking_led` revision. Do not set these revisions as current revision.



5. Update the `blinking_led_slow.qsf`, `blinking_led_empty.qsf`, `blinking_led_parent.qsf`, and `blinking_led_parent_slow.qsf` files with their corresponding `TOP_LEVEL_ENTITY` and `SYSTEMVERILOG_FILE` assignments:

```
##blinking_led_slow.qsf
set_global_assignment -name TOP_LEVEL_ENTITY blinking_led_slow
set_global_assignment -name SYSTEMVERILOG_FILE blinking_led_slow.sv
```

```
##blinking_led_empty.qsf
set_global_assignment -name TOP_LEVEL_ENTITY blinking_led_empty
set_global_assignment -name SYSTEMVERILOG_FILE blinking_led_empty.sv
```

```
##blinking_led_parent.qsf

set_global_assignment -name TOP_LEVEL_ENTITY blinking_led_parent
set_global_assignment -name SYSTEMVERILOG_FILE blinking_led_parent.sv
set_global_assignment -name SYSTEMVERILOG_FILE blinking_led.sv
```

```
##blinking_led_parent_slow.qsf

set_global_assignment -name TOP_LEVEL_ENTITY blinking_led_parent_slow
set_global_assignment -name SYSTEMVERILOG_FILE blinking_led_parent_slow.sv
set_global_assignment -name SYSTEMVERILOG_FILE blinking_led.sv
```

6. To avoid synthesis errors, ensure that the synthesis revision files for the child partitions do not contain any design partition or LogicLock Plus region assignments. Also, the synthesis revision files for the parent partitions must only contain design partition assignments for the corresponding child partitions. Remove these assignments, if any, in the `blinking_led_default.qsf`, `blinking_led_slow.qsf`, `blinking_led_empty.qsf`, `blinking_led_parent.qsf`, and `blinking_led_parent_slow.qsf` files:

```
set_instance_assignment -name PARTITION pr_partition -to
u_blinking_led_parent|u_blinking_led
set_instance_assignment -name PARTIAL_RECONFIGURATION_PARTITION ON -to
u_blinking_led_parent|u_blinking_led
set_instance_assignment -name PLACE_REGION "69 10 88 29" -to
u_blinking_led_parent|u_blinking_led
set_instance_assignment -name RESERVE_PLACE_REGION ON -to
u_blinking_led_parent|u_blinking_led
set_instance_assignment -name CORE_ONLY_PLACE_REGION ON -to
u_blinking_led_parent|u_blinking_led
set_instance_assignment -name ROUTE_REGION "68 9 89 30" -to
u_blinking_led_parent|u_blinking_led

set_instance_assignment -name PARTITION pr_parent_partition -to
u_blinking_led_parent
set_instance_assignment -name PARTIAL_RECONFIGURATION_PARTITION ON -to
u_blinking_led_parent
set_instance_assignment -name PLACE_REGION "66 7 112 32" -to
u_blinking_led_parent
set_instance_assignment -name RESERVE_PLACE_REGION ON -to
u_blinking_led_parent
set_instance_assignment -name CORE_ONLY_PLACE_REGION ON -to
```



```
u_blinking_led_parent
set_instance_assignment -name ROUTE_REGION "65 6 113 33" -to
u_blinking_led_parent
```

7. Include the following assignments in `blinking_led_parent.qsf` and `blinking_led_parent_slow.qsf` files:

```
set_instance_assignment -name PARTITION pr_partition -to u_blinking_led
set_instance_assignment -name PARTIAL_RECONFIGURATION_PARTITION ON -to
u_blinking_led
```

8. Verify that the `blinking_led.qpf` file contains the following revisions, in no particular order:

```
PROJECT_REVISION = "blinking_led"
PROJECT_REVISION = "blinking_led_pr_alpha"
PROJECT_REVISION = "blinking_led_pr_bravo"
PROJECT_REVISION = "blinking_led_pr_charlie"
PROJECT_REVISION = "blinking_led_pr_delta"
PROJECT_REVISION = "blinking_led_pr_emma"
PROJECT_REVISION = "blinking_led_default"
PROJECT_REVISION = "blinking_led_slow"
PROJECT_REVISION = "blinking_led_empty"
PROJECT_REVISION = "blinking_led_parent"
PROJECT_REVISION = "blinking_led_parent_slow"
```

*Note:* If you are copying the revision files from `hpr` folder, manually update the `blinking_led.qpf` file with the above lines of code.

## Specifying Revision Type

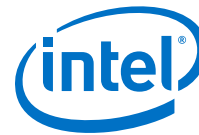
You must assign revision type for each of your revisions. There are three revision types:

- Partial Reconfiguration - Base
- Partial Reconfiguration - Persona Synthesis
- Partial Reconfiguration - Persona Implementation

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

**Table 4. Revision Types**

Revision Name	Revision Type
<code>blinking_led.qsf</code>	Partial Reconfiguration - Base
<code>blinking_led_default.qsf</code>	Partial Reconfiguration - Persona Synthesis
<code>blinking_led_empty.qsf</code>	Partial Reconfiguration - Persona Synthesis
<code>blinking_led_slow.qsf</code>	Partial Reconfiguration - Persona Synthesis
<code>blinking_led_parent.qsf</code>	Partial Reconfiguration - Persona Synthesis
<code>blinking_led_parent_slow.qsf</code>	Partial Reconfiguration - Persona Synthesis
<code>blinking_led_pr_alpha.qsf</code>	Partial Reconfiguration - Persona Implementation
<code>blinking_led_pr_bravo.qsf</code>	Partial Reconfiguration - Persona Implementation
<i>continued...</i>	



Revision Name	Revision Type
blinking_led_pr_charlie.qsf	Partial Reconfiguration - Persona Implementation
blinking_led_pr_delta.qsf	Partial Reconfiguration - Persona Implementation
blinking_led_pr_emma.qsf	Partial Reconfiguration - Persona Implementation

To specify the revision type:

1. Click **Project > Revisions**. The **Revisions** dialog box appears.
2. Select `blinking_led` in the Revision Name column, and click **Set Current**.
3. Click **Apply**. The `blinking_led` revision opens.
4. To set the revision type for `blinking_led`, click **Assignments > Settings > General**.
5. Select the **Revision Type** as **Partial Reconfiguration - Base**.
6. Similarly, set the revision types for the other ten revisions, as listed in the above table.

*Note:* You must set each revision as the current revision before assigning the revision type.

Verify that the `.qsf` file for each of the revisions contains the following assignment:

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

##blinking_led_default.qsf
set_global_assignment -name REVISION_TYPE PR_SYN

##blinking_led_slow.qsf
set_global_assignment -name REVISION_TYPE PR_SYN

##blinking_led_empty.qsf
set_global_assignment -name REVISION_TYPE PR_SYN

##blinking_led_pr_alpha.qsf
set_global_assignment -name REVISION_TYPE PR_IMPL

##blinking_led_parent.qsf
set_global_assignment -name REVISION_TYPE PR_SYN

##blinking_led_parent_slow.qsf
set_global_assignment -name REVISION_TYPE PR_SYN

##blinking_led_pr_bravo.qsf
set_global_assignment -name REVISION_TYPE PR_IMPL

##blinking_led_pr_charlie.qsf
set_global_assignment -name REVISION_TYPE PR_IMPL

##blinking_led_pr_delta.qsf
set_global_assignment -name REVISION_TYPE PR_IMPL

##blinking_led_pr_emma.qsf
set_global_assignment -name REVISION_TYPE PR_IMPL
```



**Note:** Add any Fitter specific settings that you wish to use in the PR implementation compile to the persona implementation revisions. The Fitter specific settings affect the fit of the persona, but do not affect the imported static region. You can also add any synthesis specific settings to individual persona synthesis revisions.

### Related Links

[Create Revisions for Personas](#)

## Step 8: Generating the Hierarchical Partial Reconfiguration Flow Script

To generate the hierarchical partial reconfiguration flow script:

1. From the Quartus Prime command shell, create a flow template by running the following command:

```
quartus_sh --write_flow_template -flow a10_hier_partial_reconfig
```

Quartus Prime generates the `a10_hier_partial_reconfig/flow.tcl` file.

2. Rename the generated `a10_hier_partial_reconfig/setup.tcl.example` to `a10_hier_partial_reconfig/setup.tcl`, and modify the script to specify your partial reconfiguration project details:
  - a. To define the name of the project, update the following line:

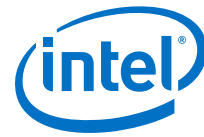
```
define_project blinking_led
```

- b. To define the base revision, update the following line:

```
define_base_revision blinking_led
```

- c. To define each of the partial reconfiguration implementation revisions, along with the PR partition names and the source revision that implements the revisions, update the following lines:

```
define_pr_impl_partition -impl_rev_name blinking_led_pr_alpha \  
-partition_name pr_partition \  
-source_rev_name blinking_led_default \  
-source_partition root_partition \  
-source_snapshot synthesized  
  
define_pr_impl_partition -impl_rev_name blinking_led_pr_alpha \  
-partition_name pr_parent_partition \  
-source_rev_name blinking_led_parent \  
-source_partition root_partition \  
-source_snapshot synthesized  
  
define_pr_impl_partition -impl_rev_name blinking_led_pr_bravo \  
-partition_name pr_partition \  
-source_rev_name blinking_led_slow \  
-source_partition root_partition \  
-source_snapshot synthesized  
  
define_pr_impl_partition -impl_rev_name blinking_led_pr_bravo \  
-partition_name pr_parent_partition \  
-source_rev_name blinking_led_pr_alpha \  
-source_partition pr_parent_partition \  
-source_snapshot final  
  
define_pr_impl_partition -impl_rev_name blinking_led_pr_charlie \  
-partition_name pr_partition \  
-source_rev_name blinking_led_slow \  
-source_partition root_partition \  
-source_snapshot synthesized
```



```

-source_rev_name blinking_led_empty \
-source_partition root_partition \
-source_snapshot synthesized

define_pr_impl_partition -impl_rev_name blinking_led_pr_charlie \
-partition_name pr_parent_partition \
-source_rev_name blinking_led_pr_alpha \
-source_partition pr_parent_partition \
-source_snapshot final

define_pr_impl_partition -impl_rev_name blinking_led_pr_delta \
-partition_name pr_partition \
-source_rev_name blinking_led_slow \
-source_partition root_partition \
-source_snapshot synthesized

define_pr_impl_partition -impl_rev_name blinking_led_pr_delta \
-partition_name pr_parent_partition \
-source_rev_name blinking_led_parent_slow \
-source_partition root_partition \
-source_snapshot synthesized

define_pr_impl_partition -impl_rev_name blinking_led_pr_emma \
-partition_name pr_partition \
-source_rev_name blinking_led_empty \
-source_partition root_partition \
-source_snapshot synthesized

define_pr_impl_partition -impl_rev_name blinking_led_pr_emma \
-partition_name pr_parent_partition \
-source_rev_name blinking_led_pr_delta \
-source_partition pr_parent_partition \
-source_snapshot final

```

*Note:* All the revision projects must be in the same directory as `blinking_led.qpf`. Otherwise, update the flow script accordingly.

## Step 9: Running the Hierarchical Partial Reconfiguration Flow Script

To run the hierarchical partial reconfiguration flow script:

1. Click **Tools** ► **Tcl Scripts**. The **Tcl Scripts** dialog box appears.
2. Click **Add to Project**, browse and select the `a10_hier_partial_reconfig/flow.tcl`.
3. Select the `a10_hier_partial_reconfig/flow.tcl` in the Libraries pane, and click **Run**.

This script runs the synthesis for the three personas. Quartus Prime generates a SRAM Object File (`.sof`), a Partial-Masked SRAM Object File (`.pmsf`), and a Raw Binary File (`.rbf`) for each of the personas.

*Note:* To run the script from the Quartus Prime command shell, type the following command:

```
quartus_sh -t a10_hier_partial_reconfig/flow.tcl -setup_script
a10_hier_partial_reconfig/setup.tcl
```

### Related Links

- [Compile the Partial Reconfiguration Design](#)
- [Using the Partial Reconfiguration Flow Script](#)
- [Configuring the Partial Reconfiguration Flow Script](#)



- [Generate Programming Files](#)

## Step 10: Programming the Board

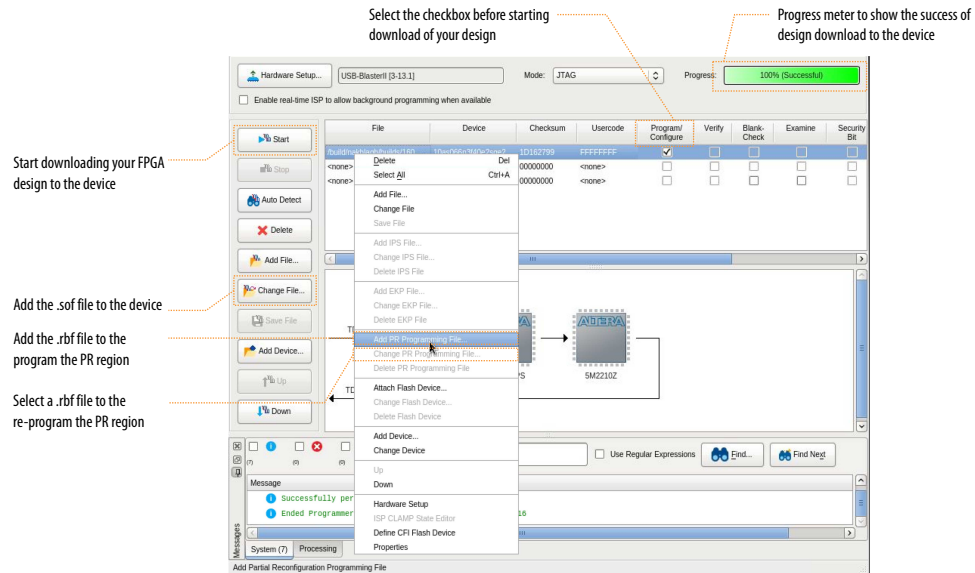
### Before you begin:

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

To run the design on the Arria 10 SoC development board:

1. Open the 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, **10AS066N3**.
4. Click **OK**. The Quartus Prime software detects and updates the Programmer with the three FPGA chips on the board.
5. Select the 10AS066N3 device, click **Change File** and load the `blinking_led_pr_alpha.sof` file.
6. Enable **Program/Configure** for `blinking_led_pr_alpha.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_pr_alpha.sof` file in the Programmer and click **Add PR Programming File**.
10. Select the `blinking_led_pr_bravo.pr_parent_partition.pr_partition.rbf` file.
11. Disable **Program/Configure** for `blinking_led_pr_alpha.sof` file.
12. Enable **Program/Configure** for `blinking_led_pr_bravo.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 Programming File**.
14. Select the `blinking_led_pr_delta.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 9. Programming the Arria 10 SoC Development Board**

## Modifying an Existing Persona

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

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

1. In the `blinking_led_slow.sv` file, modify the `COUNTER_TAP` parameter from 27 to 28.
2. To re-synthesize and re-implement this persona, you must recompile all the synthesis-only revisions and implementation revisions affected by the change. Modify the `setup.tcl` script to include the following lines:

```
define_project blinking_led

define_base_revision blinking_led

define_pr_impl_partition -impl_rev_name blinking_led_pr_bravo \
-partition_name pr_partition \
-source_rev_name blinking_led_slow \
-source_partition root_partition \
-source_snapshot synthesized

define_pr_impl_partition -impl_rev_name blinking_led_pr_bravo \
-partition_name pr_parent_partition \
-source_rev_name blinking_led_pr_alpha \
-source_partition pr_parent_partition \
-source_snapshot final

define_pr_impl_partition -impl_rev_name blinking_led_pr_delta \
-partition_name pr_partition \
-source_rev_name blinking_led_slow \
-source_partition root_partition \
-source_snapshot synthesized

define_pr_impl_partition -impl_rev_name blinking_led_pr_delta \
-partition_name pr_parent_partition \
```



```
-source_rev_name blinking_led_pr_delta \  
-source_partition pr_parent_partition \  
-source_snapshot final
```

**Note:** When defining the `pr_parent_partition` for `blinking_led_pr_delta` revision, you import the final snapshot of that persona for implementation. As a result, the implementation of the parent partition logic remains the same, while modifying and implementing the corresponding child partition.

This command re-synthesizes the `blinking_slow_led` synthesis revision, and then runs the PR implementation compile using `blinking_led_pr_bravo`.

3. To perform compilation of the synthesis-only revisions, run the following command:

```
quartus_sh -t a10_hier_partial_reconfig/flow.tcl -setup_script  
a10_hier_partial_reconfig/setup.tcl -all_syn
```

This command does not recompile the base revision.

4. To perform compilation of the implementation revisions, run the following command:

```
quartus_sh -t a10_hier_partial_reconfig/flow.tcl -setup_script  
a10_hier_partial_reconfig/setup.tcl -all_impl
```

This command does not recompile the base revision.

5. Follow the steps in [Step 10: Programming the Board](#) on page 24 to program the resulting RBF file into the FPGA.

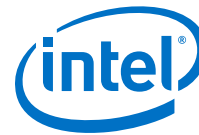
**Note:** To avoid running the entire flow for every revision, define the synthesis-only revisions and implementation revisions in the `setup.tcl` script, and run the script.

## 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 child persona for `blinking_led_parent_slow`, that turns `led_three` off:

1. Copy `blinking_led_empty.sv` to `blinking_led_off.sv`.
2. In the `blinking_led_off.sv` file, modify the assignment, assign `led_three_on = 1'b0`; to assign `led_three_on = 1'b1`. Ensure you change the module name from `blinking_led_empty` to `blinking_led_off`.
3. Create a new synthesis revision, `blinking_led_off`, by following the steps in [Creating Synthesis-Only Revisions](#) on page 18.



**Note:** The `blinking_led_off` revision must use the `blinking_led_off.sv` file.

4. Create a new implementation revision, `blinking_led_pr_foxtrot`, by following the steps in [Creating Implementation Revisions](#) on page 17.
5. Update the `a10_hier_partial_reconfig/setup.tcl` file to define the new PR implementation:

```
define_pr_impl_partition -impl_rev_name blinking_led_pr_foxtrot \
    -partition_name pr_partition \
    -source_rev_name blinking_led_off \
        -source_partition root_partition \
    -source_snapshot synthesized

define_pr_impl_partition -impl_rev_name blinking_led_pr_foxtrot \
    -partition_name pr_parent_partition \
    -source_rev_name blinking_led_pr_delta \
    -source_partition pr_parent_partition \
    -source_snapshot final
```

6. Compile just this new synthesis and implementation revision by running the following command:

```
quartus_sh -t a10_hier_partial_reconfig/flow.tcl -setup_script
a10_hier_partial_reconfig/setup.tcl -all_syn

quartus_sh -t a10_hier_partial_reconfig/flow.tcl -setup_script
a10_hier_partial_reconfig/setup.tcl -all_impl
```

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

### Related Links

- [Creating a Partial Reconfiguration Design](#)
- [Partial Reconfiguration Online Training](#)

## Document Revision History

**Table 5. Document Revision History**

Date	Version	Changes
2017.05.08	17.0.0	Initial release of the document