AN 818: Static Update Partial Reconfiguration Tutorial

for Intel® Stratix® 10 GX FPGA Development Board

Updated for Intel® Quartus® Prime Design Suite: **17.1**

Contents

1 Static Update Partial Reconfiguration Tutorial for Intel® Stratix® 10 GX FPGA Development Board

This application note demonstrates static update partial reconfiguration (SUPR) on the Intel® Stratix® 10 GX FPGA development board.

Partial reconfiguration (PR) allows you to reconfigure a portion of an Intel FPGA dynamically, while the remaining FPGA continues to operate. PR implements multiple personas in a particular region in your design, without impacting operation in areas outside this region. This methodology provides the following advantages in systems in which multiple functions time-share the same FPGA resources:

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

In traditional PR, any change to the static region requires recompilation of every persona. However, you can define a specialized SUPR region that allows change, without requiring the recompilation of personas. This technique is useful for a portion of a design that you may *possibly* want to change for risk mitigation, but that never requires runtime reconfiguration.

## 1.1 Reference Design Requirements

This reference design requires the following:

- For design implementation, the Intel Quartus® Prime Pro Edition version 17.1 with Intel Stratix 10 device support, and basic understanding of the Intel Quartus Prime design flow and project files.
- For FPGA implementation, a JTAG connection with the Intel Stratix 10 GX FPGA development board on the bench.

1.2 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 then connects the output to four LEDs on the board. Selecting the output from the counter bits, in a specific sequence, causes the LEDs to blink at a specific frequency. The top_counter module is the SUPR region.
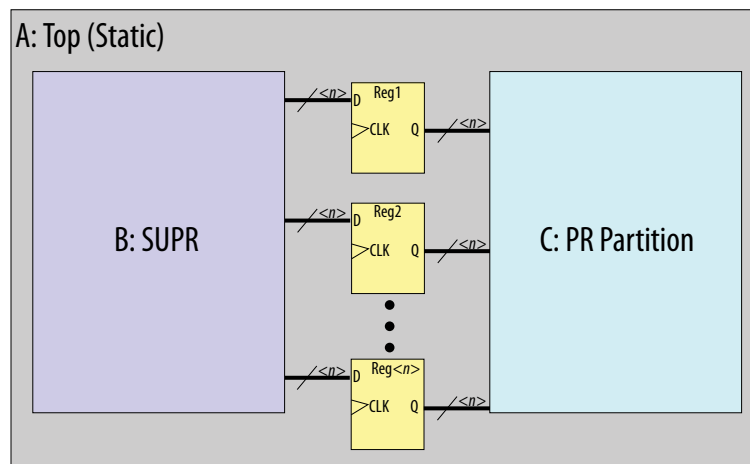
Figure 1.   Flat Reference Design
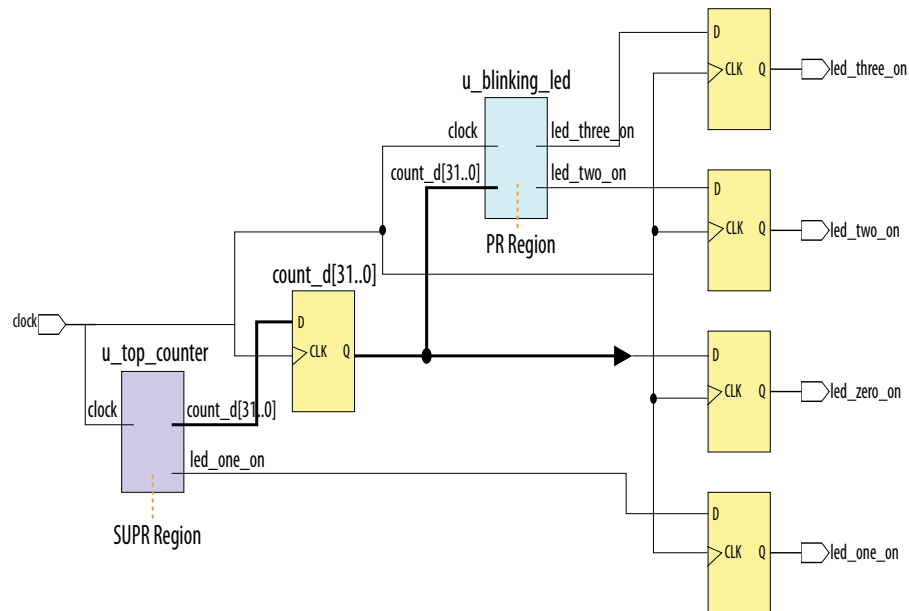
1.3 Static Update Region Overview

The following figure shows the block diagram for a PR design that includes a SUPR region. Block `A` is the Top static region. Block `B` is the SUPR region. Block `C` is the PR partition.

Figure 2.   PR Design with SUPR Region

These regions have the following characteristics:

- `A` Top Static Region—contains design logic that does not change. Changing this region requires recompilation of all associated personas. The static region includes the portion of the design that does not change for any persona. This region can include periphery and core device resources.

- `B` SUPR Region—contains core-only logic that *may* possibly change for risk mitigation, but never requires runtime reconfiguration. The SUPR region has the same requirements and restrictions as the PR partition. Changing the SUPR region produces a SRAM Object File (`.sof`) that is compatible with all existing compiled Raw Binary File (`.rbf`) files for PR partition `C`.

- `C` PR Partition—contains arbitrary logic that you can reprogram at runtime with any design logic that fits and achieves timing closure during compilation.

  Note: You must register all communication between the SUPR and PR partitions in the static region. This requirement helps to ensure timing closure for any personas, with respect to the static region.

1.4 Reference Design Walkthrough

The following steps describe implementation of SUPR with a flat design:

Note: This tutorial does not require addition of a Partial Reconfiguration Controller IP core, in contrast with *AN 817: Static Update Partial Reconfiguration Tutorial for Intel Arria® 10 FPGA Development Board*.

1.4.1 Step 1: Getting Started

To copy the reference design files to your working environment and compile the `blinking_led` flat design:
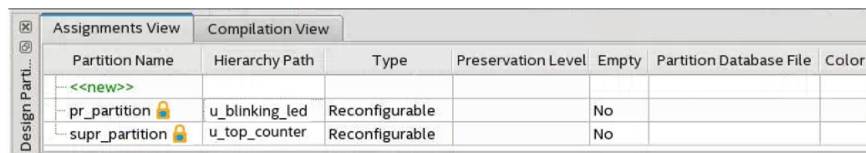
1. Create a `s10_pcie_devkit_blinking_led_supr` directory in your working environment.

2. Locate and download the `blinking_led` design example files at https:// github.com/intel/fpga-partial-reconfig.

3. Copy the downloaded `tutorials/s10_pcie_devkit_blinking_led/flat` sub-folder to the `s10_pcie_devkit_blinking_led_supr` directory.

4. In the Intel Quartus Prime Pro Edition software, click **File ➤ Open Project** and select `blinking_led.qpf`.

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

1.4.2 Step 2: Creating a Design Partition

You must create design partitions for each region that you want to partially reconfigure. You can create any number of independent partitions or PR regions in your design. The following steps describe creating design partitions for the `u_blinking_led` instance as the PR partition, and the `u_top_counter` instance as the SUPR region:

1. In the Project Navigator **Hierarchy** tab, right-click the `u_blinking_led` instance, and then click **Design Partition ➤ Set as Design Partition**. Right-click the `u_blinking_led` instance again, and then click **Design Partition ➤ Reconfigurable** under **Type**. The design partition appears on the **Assignments View** tab of the Design Partitions Window. The Intel Quartus Prime software automatically generates a partition name, based on the instance name and hierarchy path. The default partition name varies with each instance.

2. Repeat step 1 to create the partition for the `u_top_counter` instance.

3. Click **Assignments ➤ Design Partitions Window**. Double-click the **Partition Name** cell to specify the following partition names:

   - Rename `u_blinking_led` to `pr_partition`

   - Rename `u_top_counter` to `supr_partition`

Figure 3.   Design Partitions Window

Partition Name	Hierarchy Path	Type	Preservation Level	Empty	Partition Database File	Color
<<new>>						
pr_partition 🔒	u_blinking_led	Reconfigurable		No		
supr_partition 🔒	u_top_counter	Reconfigurable		No		

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

```
set_instance_assignment -name PARTITION pr_partition \
     -to u_blinking_led
set_instance_assignment -name PARTIAL_RECONFIGURATION_PARTITION ON \
     -to u_blinking_led
set_instance_assignment -name PARTITION supr_partition \
     -to u_top_counter
set_instance_assignment -name PARTIAL_RECONFIGURATION_PARTITION ON \
     -to u_top_counter
```

1.4.3 Step 3: Allocating Placement and Routing Region for a PR Partition

For every base revision that you create, the Compiler uses the PR partition region allocation to place the corresponding persona core in the reserved region. To locate and assign a PR region in the device floorplan for your base revision:

1. In the Project Navigator **Hierarchy** tab, right-click the `u_blinking_led` instance, and then click **Logic Lock Region ➤ Create New Logic Lock Region**. The region appears in the Logic Lock Regions window.

2. Your placement region must enclose the `blinking_led` logic. To select the placement region by locating the node in Chip Planner, right-click the `u_blinking_led` region name in the Logic Lock Regions window, and then click **Locate ➤ Locate in Chip Planner**.

3. Under **Partition Reports**, double-click **Report Design Partitions**. The Chip Planner highlights and color codes the region.

Figure 4.   Chip Planner Node Location for blinking_led
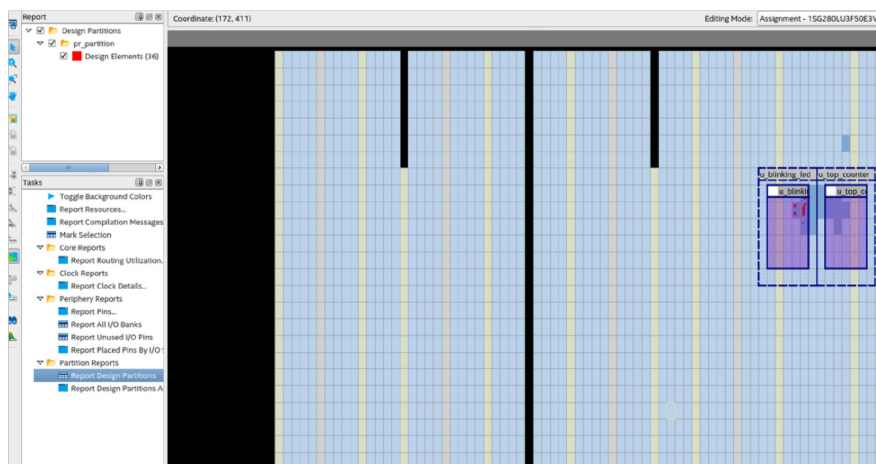
4. In the Logic Lock Regions window, specify the placement region coordinates for `u_blinking_led` in the **Origin** column. The origin corresponds to the lower-left corner of the region. For example, to set a placement region with (`X1 Y1`) co-ordinates as (`184 420`), specify the **Origin** as `X184_Y420`. The Compiler automatically calculates the (`X2 Y2`) coordinates (top-right) for the placement region, based on the height and width you specify.

5. Enable the **Reserved** and **Core-Only** options for the region.

6. Double-click the **Routing Region** option. The **Logic Lock Routing Region Settings** dialog box appears.

7. For the **Routing Type**, select **Fixed with expansion**. This option automatically assigns an **Expansion length** of one.

8. Repeat the previous steps to allocate the following resources for the `u_top_counter` partition:

   • **Origin**—`X191_Y420`

   • **Height**—5

   • **Width**—5

   Note: The routing region must be larger than the placement region, to provide extra flexibility for the Compiler's routing stage, when the Compiler routes different personas.

Figure 5. **Logic Lock Regions Window**
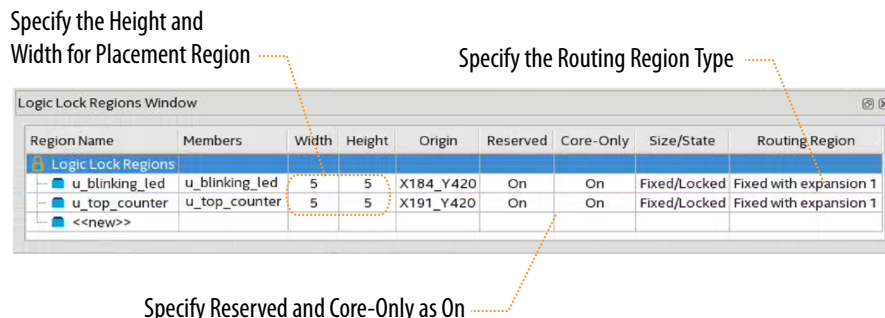
9. Verify that the `blinking_led.qsf` contains the following assignments that correspond to your floorplanning:

```
set_instance_assignment -name PARTITION supr_partition -to u_top_counter
set_instance_assignment -name PARTIAL_RECONFIGURATION_PARTITION ON -to \
    u_top_counter
set_instance_assignment -name PLACE_REGION "X191 Y420 X195 Y424" -to \
    u_top_counter
set_instance_assignment -name RESERVE_PLACE_REGION ON -to u_top_counter
set_instance_assignment -name CORE_ONLY_PLACE_REGION ON -to u_top_counter
set_instance_assignment -name ROUTE_REGION "X190 Y419 X196 Y425" -to \
    u_top_counter

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

1.4.4 Step 4: Defining Personas

This reference design defines three separate personas for the single PR partition, and one SUPR persona for the SUPR region. To define and include these personas in your project:

1. Create the following four SystemVerilog files in your working directory:

   - `blinking_led.sv`

   - `blinking_led_slow.sv`

   - `blinking_led_empty.sv`

   - `top_counter_fast.sv`

   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 1. **Reference Design Personas**

File Name	Description	Code
`blinking_led.sv`	Default persona with same design as the flat implementation	``` `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 ```
		continued...

File Name	Description	Code

		`);`
		
		`    localparam COUNTER_TAP = 23;`
		
		`    reg led_two_on_r;`
		`    reg led_three_on_r;`
		
		`    assign led_two_on = led_two_on_r;`
		`    assign led_three_on = led_three_on_r;`
		
		`    always_ff @(posedge clock) begin`
		`        led_two_on_r <= counter[COUNTER_TAP];`
		`        led_three_on_r <= counter[COUNTER_TAP];`
		`    end`
		
		`endmodule`

`blinking_led_slow.sv`	LEDs blink 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;`
		`    reg led_three_on_r;`
		
		`    assign led_two_on = led_two_on_r;`
		`    assign led_three_on = led_three_on_r;`
		
		`    always_ff @(posedge clock) begin`
		`        led_two_on_r <= counter[COUNTER_TAP];`
		`        led_three_on_r <= counter[COUNTER_TAP];`
		`    end`
		
		`endmodule`

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

`top_counter_fast.sv`	Second SUPR persona	`` `timescale 1 ps / 1 ps ``
		`` `default_nettype none ``
		
		`module top_counter_fast (`
		`    // Control signals for the LEDs`
		`    output wire led_one_on,`
		`    output wire [31:0] count,`
		`    // clock`
		`    input wire clock`
		`);`
		
		`    localparam COUNTER_TAP = 23;`
		`    reg [31:0] count_d;`
		
		`    assign count = count_d;`
		`    assign led_one_on = count_d[COUNTER_TAP];`
		
		`    always_ff @(posedge clock) begin`

continued...

File Name	Description	Code

		``
		        count_d <= count_d + 2;
		    end
		endmodule
		``

Related Links

1.4.5 Step 5: Creating Revisions

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

From the base revision, you create multiple other revisions. These other revisions contain the different implementations for the PR regions. 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, the three personas contain a base revision, four separate synthesis revisions, and four separate implementation revisions:

Table 2.   **Revisions for the Three Personas**

Synthesis Revision	Implementation Revision
`blinking_led_default`	`blinking_led_pr_alpha`
`blinking_led_slow`	`blinking_led_pr_bravo`
`blinking_led_empty`	`blinking_led_pr_charlie`
`synth_blinking_led_supr_new`	`impl_blinking_led_supr_new`

### 1.4.5.1 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 then select **blinking_led** for **Based on revision**.

4. Disable the **Set as current revision** option, and then click **OK**.

5. Similarly, repeat steps 2 through 4 to create the following revisions, based on the `blinking_led` revision:

   • `blinking_led_pr_bravo`

   • `blinking_led_pr_charlie`

   • `impl_blinking_led_supr_new`

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

Figure 6.   **Creating Revisions**
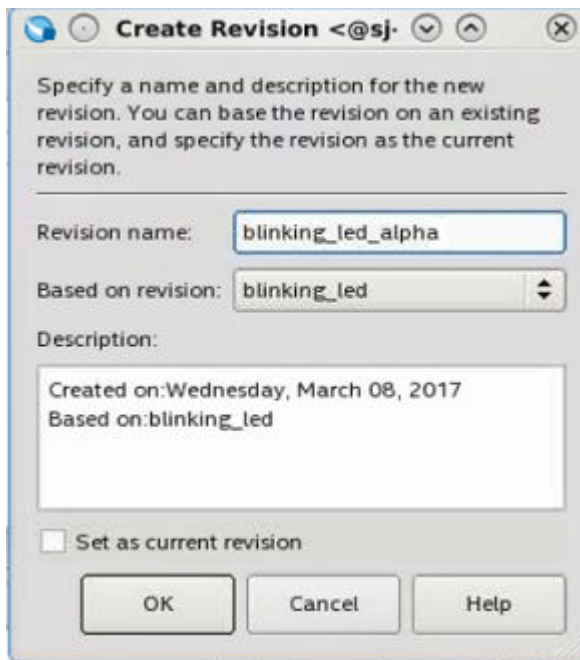
1.4.5.2 Create Synthesis-Only Revisions

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

1. In the Intel Quartus Prime software, click **Project ➤ Revisions**.

2. Create the `blinking_led_default` revision based on the `blinking_led` revision. Do not set this revision as current revision.

3. Modify the `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. Repeat steps 1 through 3 to create the following revisions. Do not set any of these revisions as current revision:

   • `blinking_led_empty`

   • `blinking_led_slow`

   • `synth_blinking_led_supr_new`

5. Update the `blinking_led_slow.qsf`, `blinking_led_empty.qsf`, and `synth_top_counter_fast.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
```

```
##synth_blinking_led_supr_new.qsf
set_global_assignment -name TOP_LEVEL_ENTITY top_counter_fast
set_global_assignment -name SYSTEMVERILOG_FILE top_counter_fast.sv
```

6. To avoid potential synthesis errors, ensure that the synthesis revision files do not contain any design partition or Logic Lock region assignments. Comment out (#) any of these assignments present in the `blinking_led_default.qsf`, `blinking_led_slow.qsf`, `blinking_led_empty.qsf`, and `synth_blinking_led_supr_new.qsf` files:

```
# set_instance_assignment -name PARTITION pr_partition \
      -to u_top_counter
# set_instance_assignment -name PARTIAL_RECONFIGURATION_PARTITION ON \
      -to u_top_counter
# set_instance_assignment -name PLACE_REGION "X191 Y420 X195 Y424" \
      -to u_top_counter
# set_instance_assignment -name RESERVE_PLACE_REGION ON \
      -to u_top_counter
# set_instance_assignment -name CORE_ONLY_PLACE_REGION ON \
      -to u_top_counter
# set_instance_assignment -name ROUTE_REGION "X190 Y419 X189 Y425" \
      -to u_top_counter

# set_instance_assignment -name PARTITION pr_partition -to u_blinking_led \
      set_instance_assignment -name PARTIAL_RECONFIGURATION_PARTITION ON \
      -to u_blinking_led
# set_instance_assignment -name PLACE_REGION "X184 Y420 X188 Y424" \
      -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 "X183 Y419 X189 Y425" \
      -to u_blinking_led
```

7. 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 = "impl_blinking_led_supr_new"
PROJECT_REVISION = "blinking_led_default"
PROJECT_REVISION = "blinking_led_slow"
PROJECT_REVISION = "blinking_led_empty"
PROJECT_REVISION = "synth_blinking_led_supr_new"
```

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

1.4.5.3 Specify the Revision Types

You must assign the 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 3.   Revision Types

Revision Name	Revision Type
`blinking_led.qsf`	Partial Reconfiguration - Base
`blinking_led_default.qsf`	Partial Reconfiguration - Persona Synthesis
`blinking_led_empty.qsf`	Partial Reconfiguration - Persona Synthesis
`blinking_led_slow.qsf`	Partial Reconfiguration - Persona Synthesis
`synth_blinking_led_supr_new.qsf`	Partial Reconfiguration - Persona Synthesis
`blinking_led_pr_alpha.qsf`	Partial Reconfiguration - Persona Implementation
`blinking_led_pr_bravo.qsf`	Partial Reconfiguration - Persona Implementation
`blinking_led_pr_charlie.qsf`	Partial Reconfiguration - Persona Implementation
`impl_blinking_led_supr_new.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 then 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. Repeat steps 2 through 5 to set the revision types for the other six revisions, as listed in the *Revision Types* table.

   Note: You must set each revision as the current revision to assign the revision type.

7. 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
```

```
##synth_blinking_led_supr_new.qsf
set_global_assignment -name REVISION_TYPE PR_SYN

##blinking_led_pr_alpha.qsf
set_global_assignment -name REVISION_TYPE PR_IMPL

##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

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

8. Add any Fitter settings to apply to the PR implementation when you compile the persona implementation revisions. Fitter settings affect the fit of the persona, but do not affect the static region that you import. Add any synthesis specific settings to individual persona synthesis revisions. Access primary Fitter and synthesis setting by clicking **Assignments ➤ Settings ➤ Compiler Settings**.

1.4.6 Step 6: Compiling the SUPR Project

SUPR requires compilation of the base revision and export of the root partition. After compilation is complete, you export the PR partitions and compile the PR regions to generate the supporting `.rbf` files. Finally, to make any change in the SUPR partition, you compile the SUPR region. The `.sof` from the updated SUPR region compilation is compatible with all previously generated `.rbf` files from the PR compiles.

Figure 7.   SUPR Compilation Flow
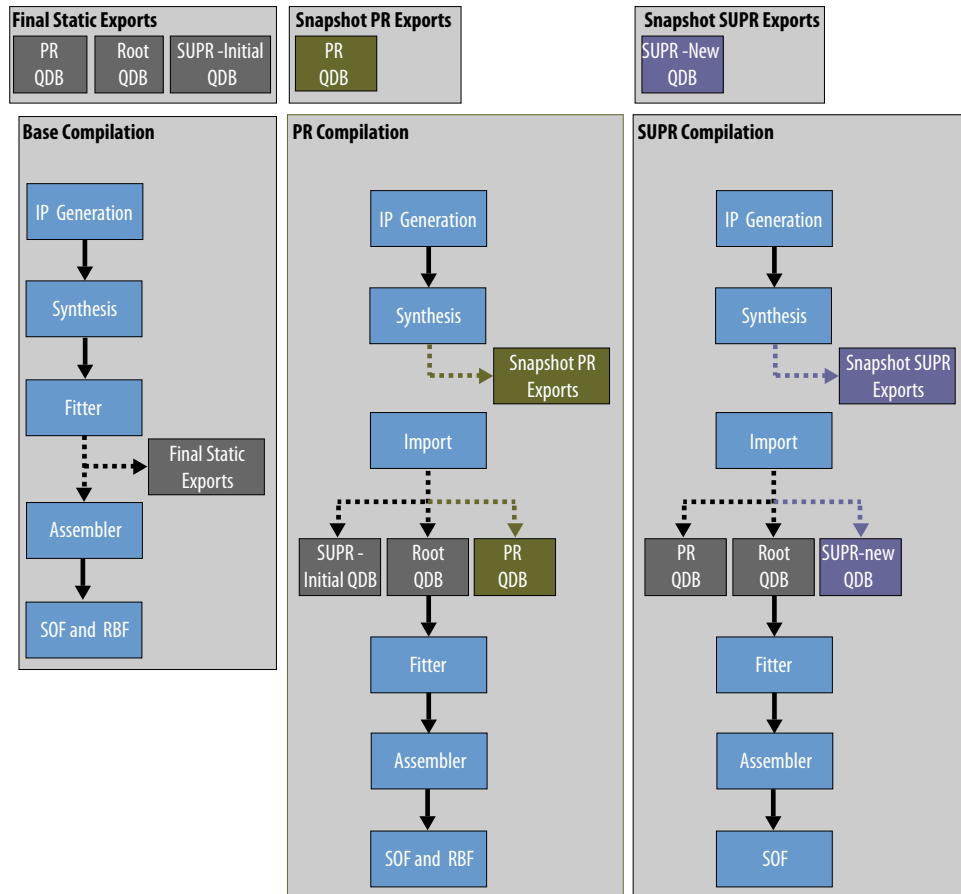
Note: The reference design includes the SUPR flow script as `flow.sh`.

1.4.6.1 Compile the Static Region

Use the following commands to compile the static region:

1. Compile the base revision:

```
quartus_ipgenerate blinking_led -c blinking_led
quartus_syn blinking_led -c blinking_led
quartus_fit blinking_led -c blinking_led
```

2. Generate the programming `.sof` and `.rbf` files:

```
quartus_asm blinking_led -c blinking_led
```

3. Export the static region, the SUPR region, and the PR region:

```
quartus_cdb blinking_led -c blinking_led --export_partition \
    root_partition --snapshot final --file \
    blinking_led.root_partition.qdb \
    --exclude_pr_subblocks

quartus_cdb blinking_led -c blinking_led --export_partition \
    supr_partition --snapshot final --file \
    blinking_led.supr_partition.qdb
```

```
quartus_cdb blinking_led -c blinking_led --export_partition \
     pr_partition --snapshot final --file \
     blinking_led.pr_partition.qdb
```

1.4.6.2 Compile the PR Region

Use the following commands to compile the PR region:

1. Synthesize the default PR persona:

   ```
   quartus_ipgenerate blinking_led -c blinking_led \
        quartus_syn blinking_led -c blinking_led_default
   ```

2. Export the persona:

   ```
   quartus_cdb blinking_led -c blinking_led_default --export_partition \
        root_partition --snapshot synthesized --file blinking_led_default.qdb
   ```

3. Import the static, SUPR, and PR blocks:

   ```
   quartus_cdb blinking_led -c blinking_led_pr_alpha --import_partition \
        root_partition --file blinking_led.root_partition.qdb

   quartus_cdb blinking_led -c blinking_led_pr_alpha --import_partition \
        supr_partition --file blinking_led.supr_partition.qdb

   quartus_cdb blinking_led -c blinking_led_pr_alpha --import_partition \
        pr_partition --file blinking_led_default.qdb
   ```

4. Run the Fitter:

   ```
   quartus_fit blinking_led -c blinking_led_pr_alpha
   ```

5. Generate the programming `.sof` and `.rbf` files:

   ```
   quartus_asm blinking_led -c blinking_led_pr_alpha
   ```

6. Repeat steps 1 through 5 for the remaining slow and empty personas in the bravo and charlie implementation, respectively.

1.4.6.3 Compile the SUPR Region

To make any changes in the SUPR region, you must recompile with the new SUPR region and generate a new base `.sof` device configuration file. Then, reuse the block in your design without recompiling the personas. The `.sof` that you generate is compatible with all the previously generated PR personas.

1. Synthesize the SUPR persona:

   ```
   quartus_syn blinking_led -c synth_blinking_led_supr_new
   ```

2. Export the SUPR persona:

   ```
   quartus_cdb blinking_led -c synth_blinking_led_supr_new \
        --export_partition root_partition --snapshot synthesized --file
   ```

3. Import the static, SUPR, and PR blocks:

   ```
   quartus_cdb blinking_led -c impl_blinking_led_supr_new --import_partition \
        root_partition --file blinking_led.root_partition.qdb

   quartus_cdb blinking_led -c impl_blinking_led_supr_new --import_partition \
        supr_partition --file synth_blinking_led_supr_new.supr_partition.qdb
   ```
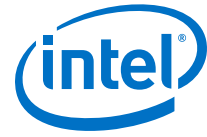
```
quartus_cdb blinking_led -c impl_blinking_led_supr_new --import_partition \
      pr_partition --file blinking_led.pr_partition.qdb
```

4. Run the Fitter:

```
quartus_fit blinking_led -c impl_blinking_led_supr_new
```

5. To generate the `.sof` and `.rbf` files, run the Assembler:

```
quartus_asm blinking_led -c impl_blinking_led_supr_new
```

1.4.7 Step 7: Programming the Board

Follow these steps to connect and program the Intel Stratix 10 GX FPGA development board:

1. Connect the power supply to the Intel Stratix 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.

3. Open the Intel Quartus Prime software, and then click **Tools ➤ Programmer**.

4. In the Programmer, click **Hardware Setup**, and then select **USB-Blaster**.

5. Click **Auto Detect**, and then select the **1SG280LU5S1** device.

6. Click **OK**. The Intel Quartus Prime software detects and updates the Programmer with the three FPGA devices on the board.

7. Select the **1SG280LU5S1** device, click **Change File**, and load the `blinking_led_pr_alpha.sof` file.

8. Enable **Program/Configure** for the `blinking_led_pr_alpha.sof` file.

9. Click **Start** and wait for the progress bar to reach 100%.

10. Observe the LEDs on the board blinking at the same frequency as the original flat design.

11. To program only the PR region, right-click the `blinking_led_pr_alpha.sof` file in the Programmer and click **Add PR Programming File**.

12. Select the `blinking_led_pr_bravo.rbf` file.

13. Disable **Program/Configure** for the `blinking_led_pr_alpha.sof` file.

14. Enable **Program/Configure** for the `blinking_led_pr_bravo.rbf` file, and then click **Start**. On the board, observe `LED[0]` and `LED[1]` continuing to blink. When the progress bar reaches 100%, `LED[2]` and `LED[3]` blink slower.

15. To re-program the PR region, right-click the `.rbf` file in the Programmer, and then click **Change PR Programing File**.

16. Select the `.rbf` files for the other two personas to observe the behavior on the board. Loading the `blinking_led_pr_alpha.rbf` file causes the LEDs to blink at a specific frequency, and loading the `blinking_led_pr_charlie.rbf` file causes the LEDs to stay ON.

Figure 8.   Programming the Intel Stratix 10 GX FPGA Development Board
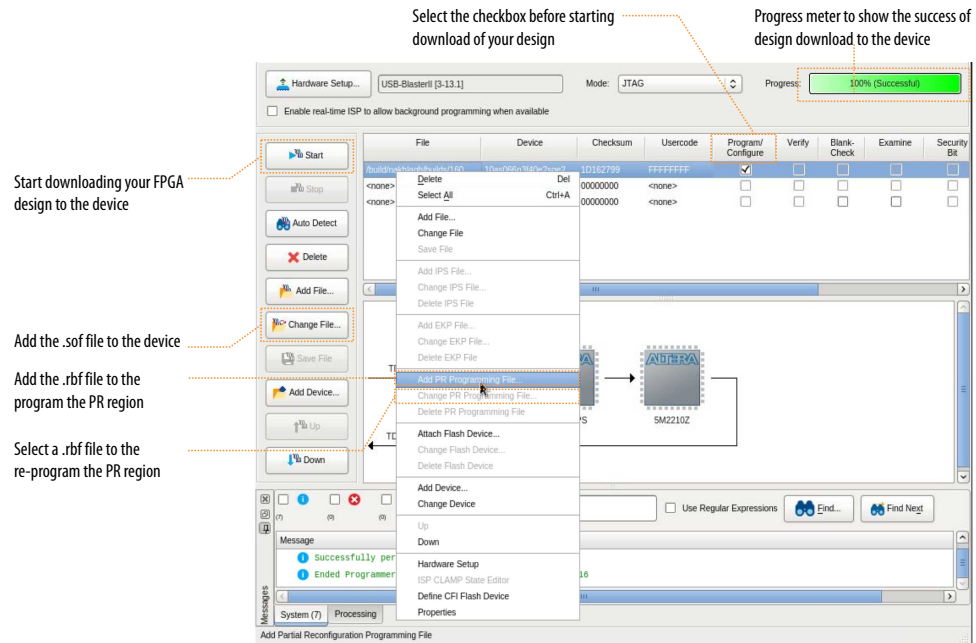
1.4.8 Modifying the SUPR Block

You can modify an existing SUPR block. After modifying the SUPR block, you must compile it, generate the `.sof` file, and program the board, without compiling the other personas. For example, follow these steps to change the `top_counter_fast.sv` module to count faster:

1. In the `top_counter_fast.sv` file, replace the `count_d + 2` statement with `count_d + 4`.

2. Run the following commands to re-synthesize the SUPR block and generate the new `.sof` file:

```
# re-synthesize SUPR persona
quartus_syn blinking_led -c synth_blinking_led_supr_new

# Export SUPR persona
quartus_cdb blinking_led -c synth_blinking_led_supr_new \
    --export_partition root_partition --snapshot synthesized --file

# Import Blocks (SUPR, Static, PR)
quartus_cdb blinking_led -c impl_blinking_led_supr_new --import_partition \
    root_partition --file blinking_led.root_partition.qdb

quartus_cdb blinking_led -c impl_blinking_led_supr_new --import_partition \
    supr_partition --file synth_blinking_led_supr_new.supr_partition.qdb

quartus_cdb blinking_led -c impl_blinking_led_supr_new --import_partition \
    pr_partition --file blinking_led.pr_partition.qdb

# Run fitter
quartus_fit blinking_led -c impl_blinking_led_supr_new

# ASM
quartus_asm blinking_led -c impl_blinking_led_supr_new
```

The resulting `.sof` now contains the new SUPR region, and uses `blinking_led` for the default (power-on) persona.

Note: The Assembler generates an `.rbf` file for the SUPR region. However, you cannot use this file to reprogram the FPGA at runtime. Ignore this `.rbf` file.

For complete information on partially reconfiguring your design for Intel Stratix 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

1.5 Document Revision History

Table 4.   Document Revision History

Document Version	Software Version	Changes
2017.11.06	17.1.0	Initial release of the document.