Intro to Partial Reconfiguration Flow Lab

Introduction

In this lab, you are dealing with a design that has two Reconfigurable Partitions (RP), each having two Reconfigurable Module (RM). The design source consists of HDL (Verilog and VHDL) files. You will use Vivado with Partial Reconfiguration (PR) capability enabled to synthesize HDL models and implement the design. You will go through the PR flow to generate the full and partial bitstreams.

Objectives

After completing this lab, you will be able to:

- Use Tcl script to generate a Vivado IPI design, create a wrapper file from it, add the static logic files and generate the design checkpoint
- Use Vivado's bottom-up methodology to synthesize the necessary RMs
- Floorplan the design
- Add the desired RMs
- Create multiple configurations
- Implement the design and generate full and partial bitstreams for various configurations
- Download bitstreams to demonstrate a working partial reconfigurable design

Design Description

The purpose of this lab exercise is to implement a design that is dynamically reconfigurable using Vivado. The design, shown in Figure 2, consists of two RP, each having two RM. The two RP are called math and led. The math RP consists of two functions: addition and subtraction, whereas the led RP consist of right and left shifting pattern of LEDs. User interacts with math RP using a terminal emulator program whereas interaction with led RP is achieved using push-buttons. The dynamic modules are downloaded using the Vivado Hardware Manager.

The directory structure of the project is shown below. The home directory of the lab contains some of the Tcl scripts which you will use to generate the initial design and then to synthesize the RMs. The Bitstreams, Checkpoint, Implement, and Synth directories are place holders where intermediate files are stored as you progress through the lab. The Sources directory consists of several sub-folders. The BOOT directory has the BOOT.bin file which is needed for the board to boot from.

The provided design places the UART (RX and TX) pins of the PS (Processing System) on the Cortex-A9 in a simple GPIO mode to allow the UART to be connected (passed through) to the Programmable Logic. The processor samples the RX signal and sends it to the EMIO channel 1 which is connected to Rx input of the HDL module provided in the Static directory. Similarly, the design samples the Tx output of the HDL module through another EMIO channel and sends it on the PS UART TX pin. This is done through a software application provided in the uart_through_gpio.sdk folder hierarchy. The design is shown in Figure 2.

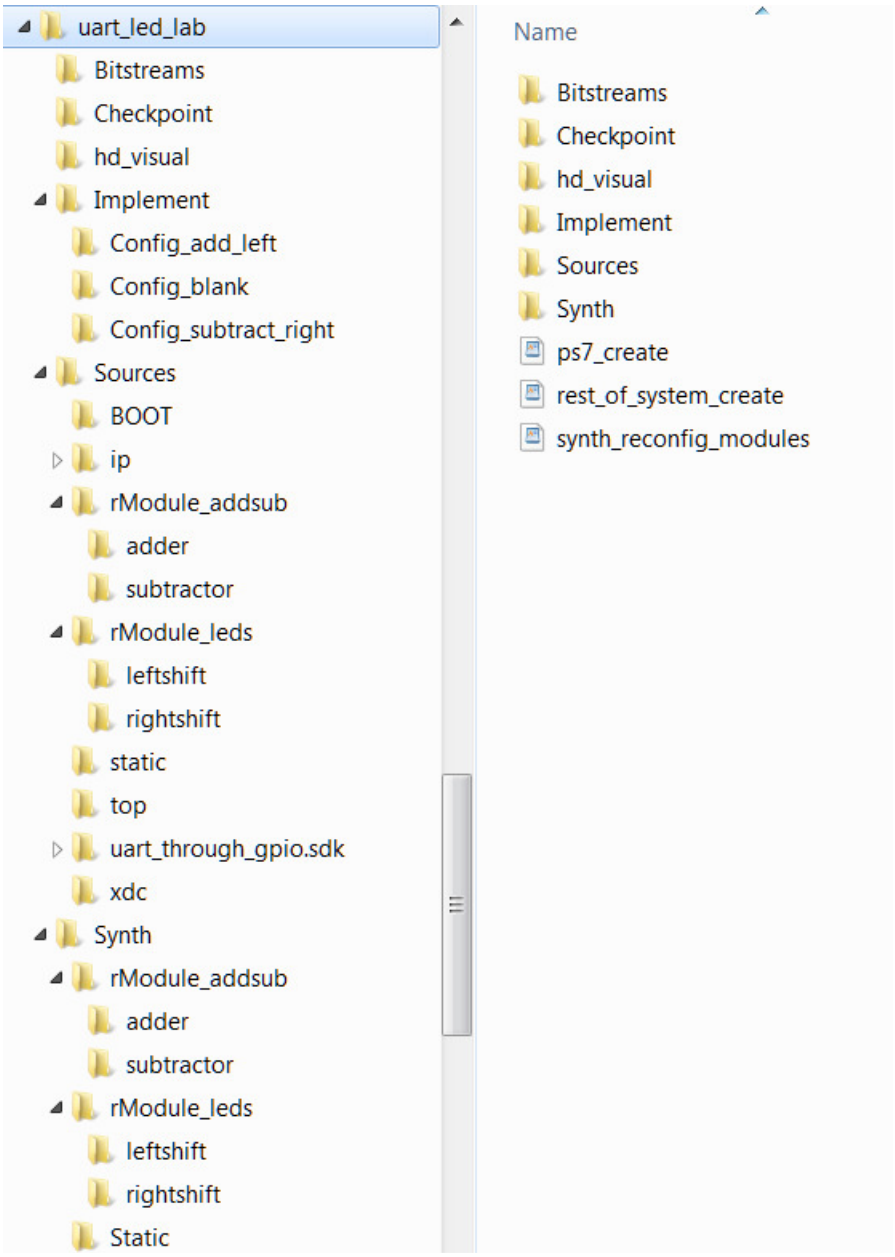


Figure 1. Directory structure of the lab

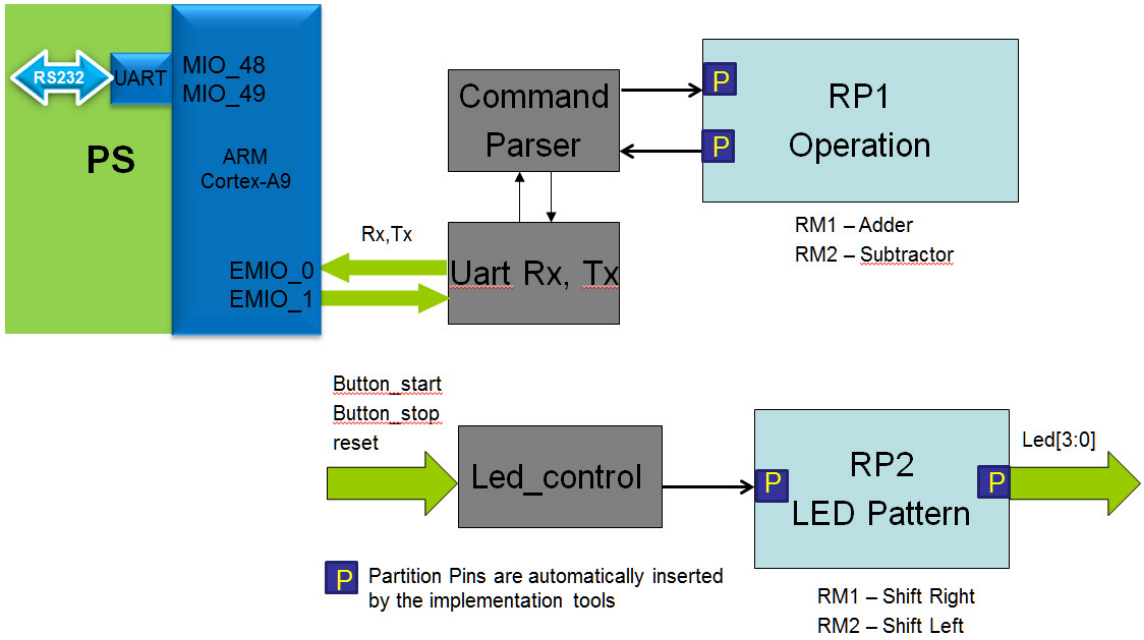
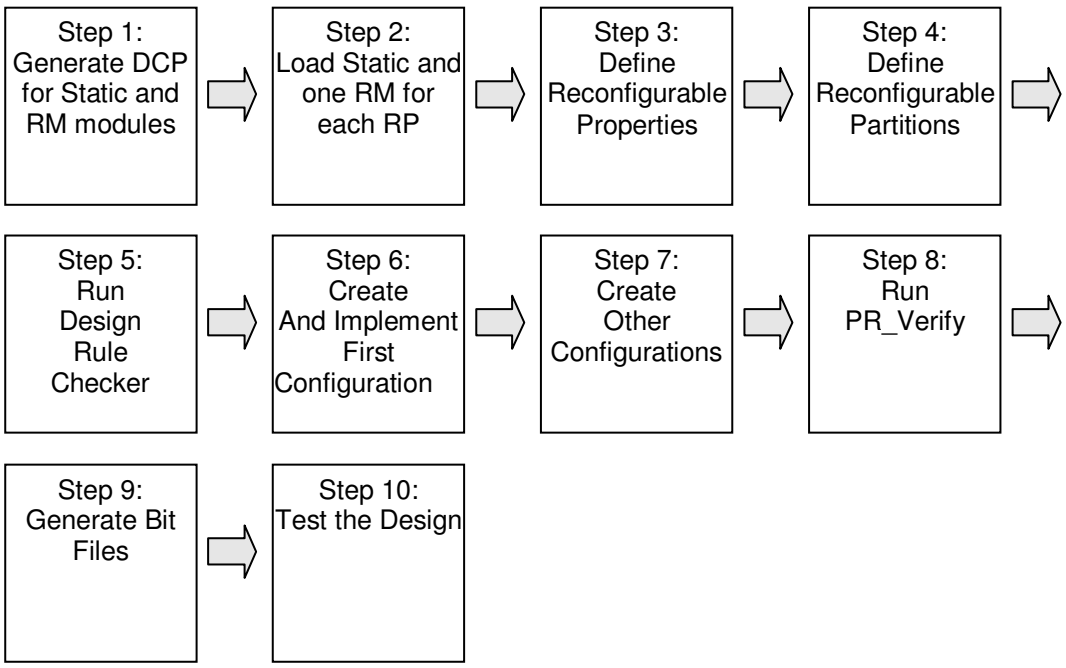


Figure 2. A Complete System

Procedure

This lab is separated into steps that consist of general overview statements that provide information on the detailed instructions that follow. Follow these detailed instructions to progress through the lab.

General Flow for this Lab



Generate DCPs for the Static Design and RM Modules Step 1

In this design we will use Zybo's USB-UART which is controlled by the Zynq's ARM Cortex-A9 processor. Our PL design needs access to this USB-UART. So first thing we will do is to create a Processing System design which will put the USB-UART connections in a simple GPIO-style and make it available to the PL section.

1-1. Start Vivado and execute the provided Tcl scripts to create the design check point for the static design. The design has two RPs.

1-1-1. Open **Vivado** by selecting **Start > All Programs > Xilinx Design Tools > Vivado 2014.3 > Vivado 2014.3**

On Linux machine, open a terminal window and then type **vivado.**

1-1-2. In the Tcl Shell window enter the following command to change to the lab directory and hit **Enter**.

```
cd c:/xup/PR/labs/uart_led_lab
```

1-1-3. Generate the PS design by executing the provided Tcl script.

```
source ps7_create.tcl
```

This script will create a block design called *system*, instantiate ZYNQ PS with SD 0 interface enabled, connect UART 1 to GPIO channels 48 and 49, enable FCLK0, RESET0_N, and two EMIO channels. It will then create a top-level wrapper file called system_wrapper.v which will instantiate the system.bd (the block design). You can check the contents of the tcl files to confirm the commands that are being run.

1-1-4. Execute the provided rest_of_system_create.tcl file.

```
source rest_of_system_create.tcl
```

This will generate the rest of the static design, include the generated PS system, and synthesize the entire design, generating the design check point (**top_wrapper.dcp**) file in the **uart_led_lab/uart_led_lab.runs/synth_1** directory.

Wait for the synthesis process to complete. When complete, the project will close

1-1-5. Using the Windows Explorer, copy the **top_wrapper.dcp** file from c:\xup\PR\labs \uart_led_lab\uart_led_lab\uart_led_lab.runs\synth_1 into the *Synth**Static* directory under the current lab directory.

1-2. Since we have RMs in HDL format, we need to synthesize them and generate the dcp for each of the RMs. The generated DCPs should be stored in appropriate directories so they can be accessed correctly; particularly, the dcp files for RM must be in separate directories as their dcp file names will be same for a given RP.

1-2-1. In the Tcl Shell window make sure that the working directory is the lab root directory. If not then enter the following command to change to the lab root directory and hit **Enter**.

```
cd c:/xup/PR/labs/uart_led_lab
```

1-2-2. Synthesize each of the RMs (two for addsub and two for the shift) by executing the provided Tcl script.

```
source synth_reconfig_modules.tcl
```

This script will add the HDL files (the *addsub* modules are in VHDL whereas the *shift* modules are in Verilog) for a given RM, synthesize the module(s) in out of context mode and write the design checkpoint (dcp) in the respective destination folder under the Synth directory. After each RM's dcp is generated, the respective design is closed.

1-2-3. At this point the directory content will look like shown below.

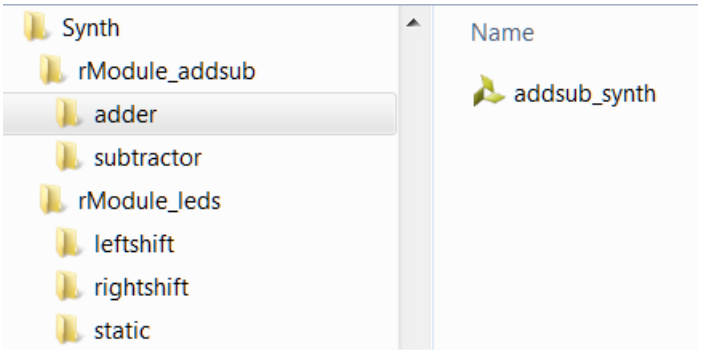


Figure 3. Synth directory hierarchy and content

Load Static and one RM for each RPs in Vivado Step 2

Since all required netlist files (dcp) for the design are now available, you will use Vivado to floorplan the design, define Reconfigurable Partitions, add Reconfigurable Modules, run the implementation tools, and generate the full and partial bitstreams.

2-1. In this step you will load the static design and one RM design for each of the RPs.

2-1-1. In the Tcl Shell window make sure that the working directory is the lab root directory otherwise execute the following command to change to the lab directory and hit **Enter**.

```
cd c:/xup/PR/labs/uart_led_lab
```

2-1-2. Load the static design using the **open_checkpoint** command.

```
open_checkpoint Synth/Static/top_wrapper.dcp
```

You can see the design structure in the Netlist pane with two black boxes for the **reconfig_addsub** and **reconfig_leds** modules.

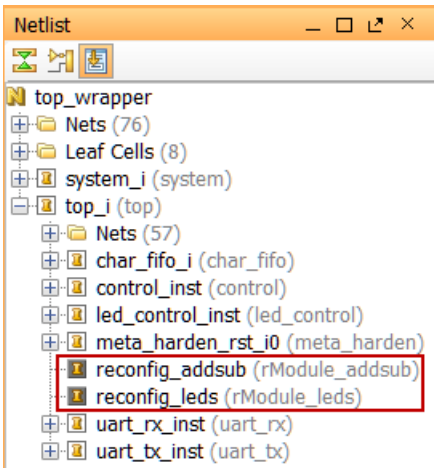


Figure 4. Static design with two black boxes

Two critical warnings are issued regarding unmatched instances. These warnings are caused because the reconfigurable modules have yet to be loaded, and can safely be ignored.

2-1-3. Set the HD_VISUAL parameter ON so we can see the frames of the actual implemented pblocks later on by executing the following Tcl command.

```
set_param hd.visual 1
```

2-1-4. Select the *reconfig_addsub* instance and then select the *Properties* tab in the **Cell Properties** window. Note that the *IS_BLACKBOX* checkbox is checked.

2-1-5. Load one RM for each RP by using the **read_checkpoint** command.

```
read_checkpoint -cell top_i/reconfig_addsub
Synth/rModule_addsub/adder/addsub_synth.dcp

read_checkpoint -cell top_i/reconfig_leds
Synth/rModule_leds/leftshift/shift_synth.dcp
```

You can now see the design structure in the Netlist pane with an RM for the *reconfig_addsub* and *reconfig_leds* modules loaded.

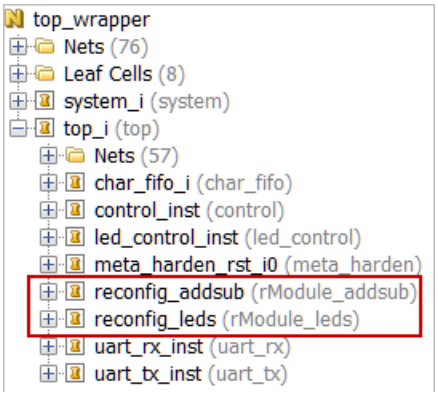


Figure 5. Static design with RM loaded

2-1-6. Select the *reconfig_addsub* instance and then select the *Properties* tab in the **Cell Properties** window. Note that the *IS_BLACKBOX* checkbox is not checked since a RM design is loaded.

2-1-7. Select the *Statistics* tab and note the amount and type of resources the module uses.

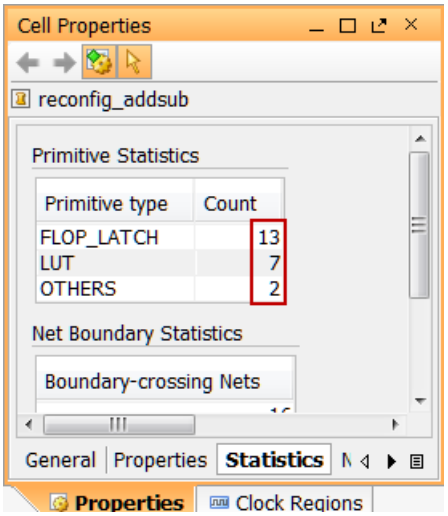


Figure 6. Resources used by the loaded RM

2-1-8. Similarly, you can select the *reconfig_leds* instance and then select the *Statistics* tab to note the type and amount of resources used.

Define Reconfigurable Properties on each RM Step 3

3-1. In this design you have two Reconfigurable Partitions each having two RMs. Define the reconfigurable properties to each of the loaded RMs.

3-1-1. Define each of the loaded RMs (submodules) as partially reconfigurable by setting the **HD.RECONFIGURABLE** property using the following commands.

```
set_property HD.RECONFIGURABLE 1 [get_cells top_i/reconfig_addsub]

set_property HD.RECONFIGURABLE 1 [get_cells top_i/reconfig_leds]
```

This is the point at which the Partial Reconfiguration license is checked. If you have a valid license, you see this message:

Feature available: PartialReconfiguration

3-1-2. Select the `reconfig_addsub` instance and notice that the DONT_TOUCH checkbox is selected in the **Cell Properties** window.

3-1-3. Save the assembled design state for this initial configuration (Is this required or optional) using the following command.

```
write_checkpoint Checkpoint/top_link_add_left.dcp
```

This will write the *dcp* file in the provided **Checkpoint** directory.

Define the Reconfigurable Partition Region Step 4

4-1. Next you must floorplan the RP regions. Depending on the type and amount of resources used by each RM, the RP region must be appropriately defined so it can accommodate any RM variant.

4-1-1. Select **Edit > Find**. In the *Find* field. Select **Sites** in the *Find* drop-down box.

4-1-2. Ensure *Name* and *contains* are selected, and in the text box change * to ***SLICE_X24Y36**.

4-1-3. Click on the **+** button, then select **OR** using the drop-down button, choose *Name contains* again, type ***SLICE_X27Y39**, and click **OK**.

You will see a new tab, called Sites – Find will appear showing two entries.

4-1-4. Select one entry at a time, right-click and select **Mark**.

You will see marked sites in the Device window. You may have to zoom out.

4-1-5. Select the **reconfig_addsub** instance in the *Netlist* window, right-click, and select *Floorplanning > Draw Pblock.*

4-1-6. Draw a box that bounds `SLICE_X22Y36:SLICE_X25Y39` marked in the previous step.

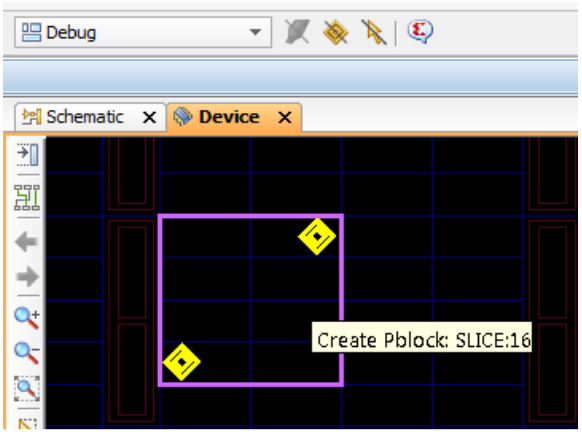


Figure 7. Drawing the addsub Pblock

4-1-7. Click **OK** to include SLICE as the resources to be reconfigured.

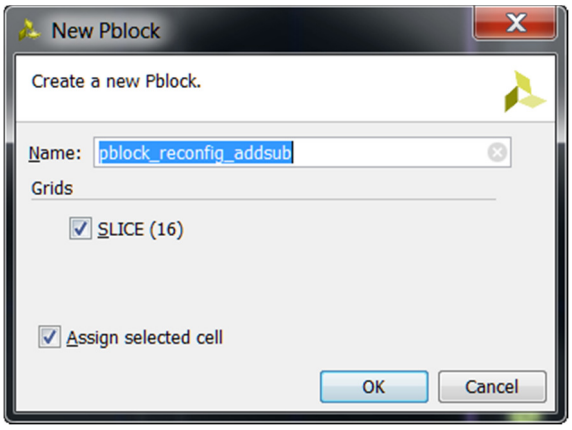


Figure 8. Defining resources to reconfigure

4-1-8. Similarly, mark the ***SLICE_X22Y20** and ***SLICE_X25Y24** sites.

4-1-9. Select the **reconfig_leds** instance in the **Netlist** window, right-click, and select *Floorplanning > Draw Pblock.*

4-1-10. Draw a box that bounds `SLICE_X22Y20:SLICE_X25Y24` (20 slices).

4-1-11. Click **OK.**

Run Design Rule Checker Step 5

5-1. It is always good idea to run a design rule checker so you can catch errors as soon as possible.

5-1-1. Select **Tools > Report > Report DRC.**

5-1-2. Deselect **All Rules**, select **Partial Reconfiguration**, and then click **OK** to run the PR-specific design rules.

You will see one error and one warning. The first error indicates that we have two RMs in the same column which is not permitted. Partial bitstreams by construction are the height of a full clock region, so Reconfigurable Partitions may not reside above or below each other in the same clock region. One of these two pblocks must be moved to the left or right so no vertical overlap occurs.

The warning hints: “4 cells with INIT values are found without resets inside the Pblock 'pblock_reconfig_leds'. Without a reset the INIT value will not be loaded during a partial reconfiguration. To fix this issue do one of the following: 1) set the Pblock property RESET_AFTER_RECONFIG=TRUE on the Pblock. Using this constraint requires that the Pblock RANGEs be frame aligned. 2) add a reset to each cell that can be held during and released after a partial reconfiguration.”

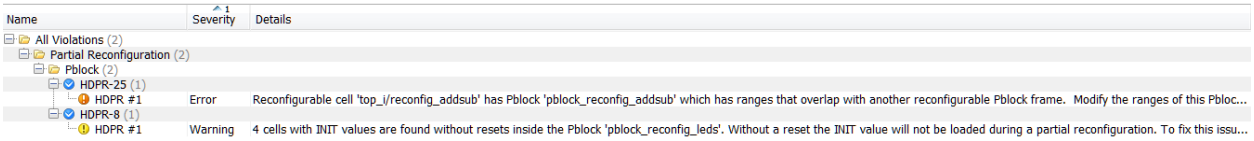


Figure 9. DRC output

5-1-3. Using the Pblock for **reconfig_leds**, drag and drop it left so that it does not fall under the other pblock and is positioned to *SLICE_X16Y20:SLICE_X19Y24*.

You can use the Find command, search for SLICE_X16Y20, mark it and then grab and drop the pblock to the left of the marked site.

5-1-4. Select **Tools > Report > Report DRC,** then click **OK** to run the PR-specific design rules.

You will notice that one warning related to **reconfig_leds** remains**.**

5-1-5. Select the **reconfig_leds** instance, select the *properties* tab, check the *RESET_AFTER_RECONFIG* property, and select **ON** for the *SNAPPING_ON* property.

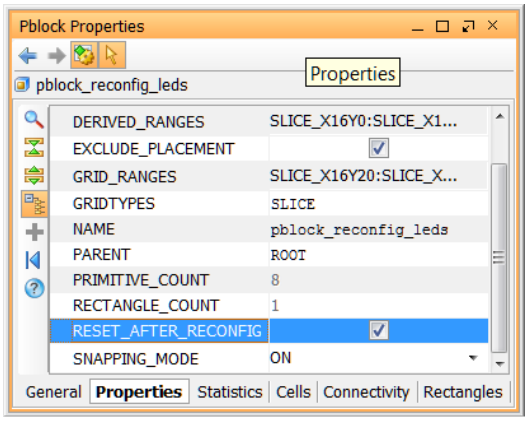


Figure 10. Enabling RESET_AFTER_RECONFIG and turning ON SNAPPING_MODE

5-1-6. Select **Tools > Report > Report DRC,** then click **OK** to run the PR-specific design rules. There should be no violations.

5-1-7. Save the Pblocks and associated properties by issuing the following command.

```
write_xdc Sources/xdc/fplan.xdc
```

Create and Implement First Configuration — Step 6

6-1. Add the top-level io and static design constraints. Create and implement the first Configuration.

6-1-1. Load the top-level constraint file by executing the following command.

```
read_xdc Sources/xdc/top_io.xdc
```

This sets the device pinout and top-level timing constraints. This top-level XDC file should only contain constraints that reference objects in the static design. Constraints for logic or nets inside of the RP can be applied for specific Reconfigurable Modules if needed.

6-1-2. Optimize, place and route the design by executing the following commands.

```
opt_design

place_design

route_design
```

6-1-3. Once the design is implemented (placed and routed), zoom in into the Device view to see the *pblock_reconfig_addsub*.

You will see the introduction of Partition Pins. These are the physical interface points between static and reconfigurable logic and are the replacement in Vivado for what was Proxy Logic in ISE, but without the requirement of LUT1 insertion. They are anchor points within an interconnect tile through which each IO of the reconfigurable module must route. They appear as white boxes in the placed design view.

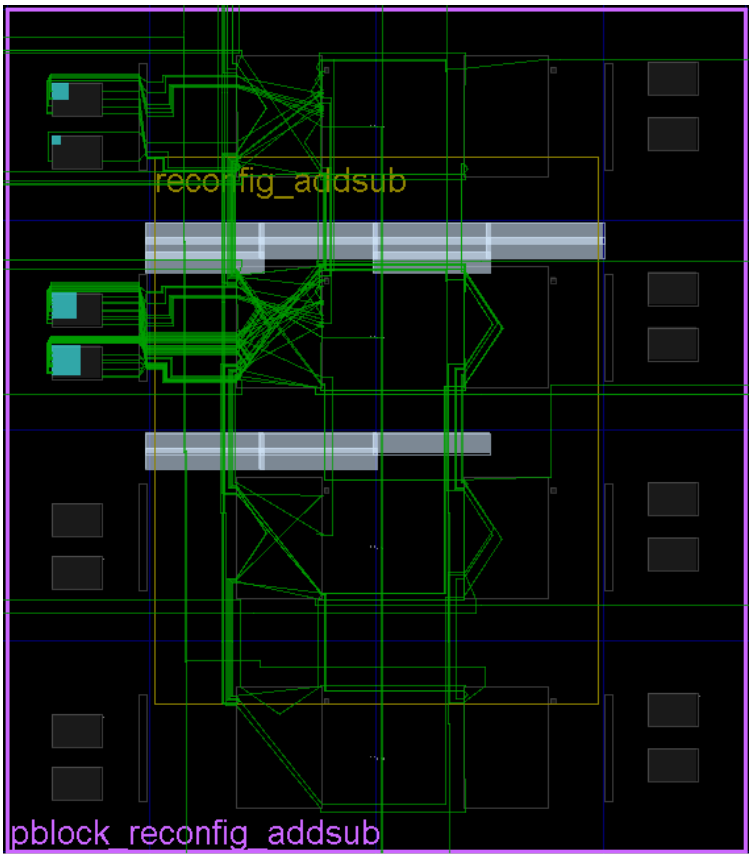


Figure 11. Partition pins in pblock_reconfig_addsub

6-1-4. Select one of the pins and see the **Cell Pins** tab of the *Partition Pin Properties*.

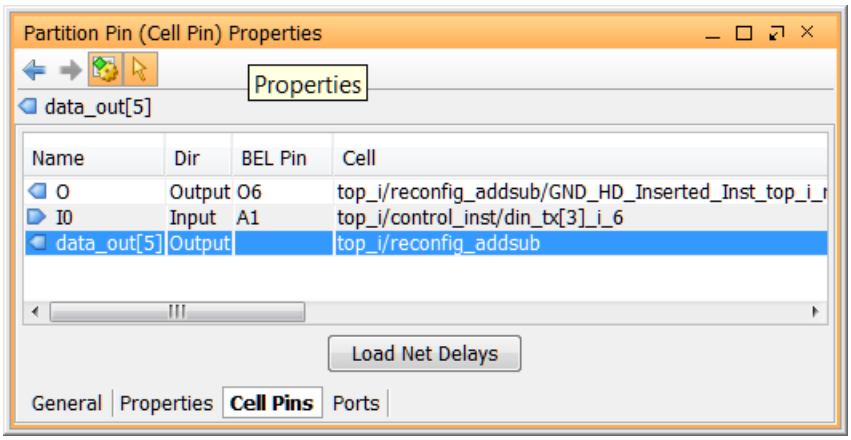


Figure 12. Partition pin properties in the Cell Pins tab

6-1-5. Now view the *pblock_reconfig_leds* block area and notice that there are no partition pins placed inside the defined pblock. If you zoom out more, you will find its partition pins located towards the top-edge of the clock region (still in the same column area). This is due to SNAPPING_MODE property we had turned ON earlier. With that property turned ON the whole column going through the drawn pblock is covered by the given pblock and no static logic will be placed there.

6-1-6. Verify that by executing the following to Tcl commands:

source ./hd_visual/pblock_reconfig_leds_AllTiles.tcl

highlight_objects -color yellow [get_selected_objects]

Many Tcl scripts are generated under hd_visual directory since we had turned ON the HD VISAUL property.

If you zoom out and or resize the viewing area, you will see the highlighted frames. If you zoom in enough you will notice the actual defined pblock area.

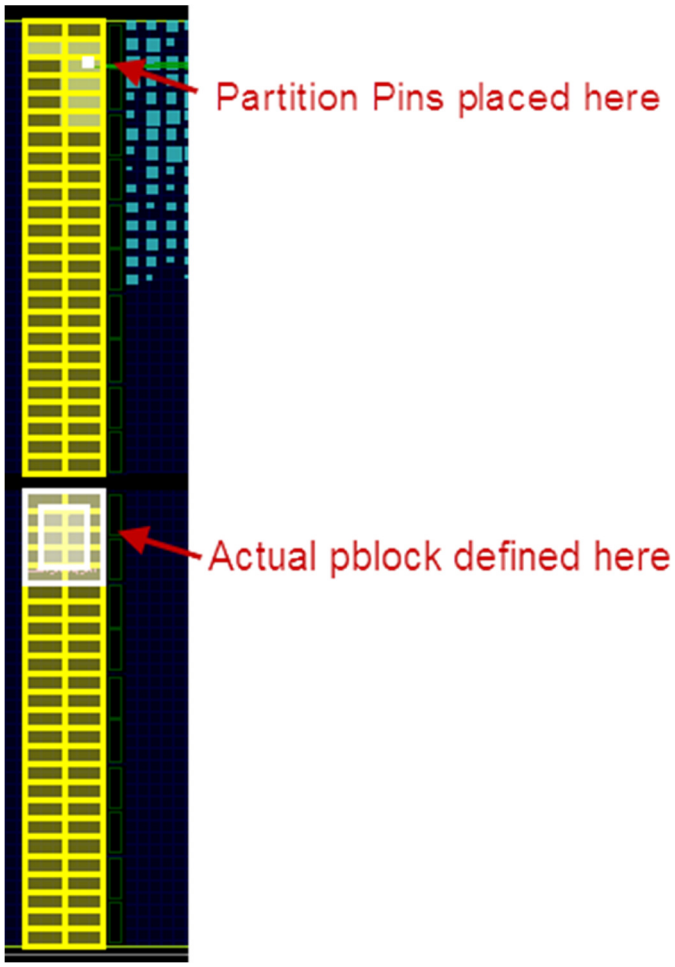


Figure 13. Reconfigurable partition frames are highlighted

6-1-7. Save the full design checkpoint and create report files by executing the following commands:

```
write_checkpoint -force Implement/Config_add_left/top_route_design.dcp

report_utilization -file Implement/Config_add_left/top_utilization.rpt

report_timing_summary -file
Implement/Config_add_left/top_timing_summary.rpt
```

6-1-8. Save checkpoints for each of the reconfigurable modules by issuing these two commands:

```
write_checkpoint -force -cell top_i/reconfig_addsub
Checkpoint/addsub_add_route_design.dcp

write_checkpoint -force -cell top_i/reconfig_leds
Checkpoint/shift_left_route_design.dcp
```

At this point, a fully implemented partial reconfiguration design from which full and partial bitstreams can be generated is ready. The static portion of this configuration **must** be used for all subsequent configurations, and to isolate the static design, the current reconfigurable modules must be removed.

6-2. After the first configuration is created, the static logic implementation will be reused for the rest of the configurations. So it should be saved. But before you save it, the loaded RM should be removed.

6-2-1. Clear out the existing RMs executing the following commands.

```
update_design -cell top_i/reconfig_addsub -black_box

update_design -cell top_i/reconfig_leds -black_box
```

Issuing these commands will result in design changes including, the number of Fully Routed nets (green) being decreased, the number of Partially Routed nets (yellow) being increased, and *inst_shift* and *inst_count* will appear in the Netlist view as empty.

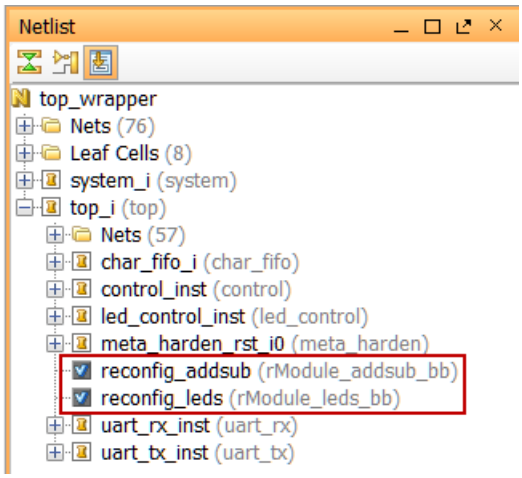


Figure 14. The design with unloaded modules

6-2-2. Lock down all placement and routing by executing the following command.

```
lock_design -level routing
```

Because no cell was identified in the `lock_design` command, the entire design in memory (currently consisting of the static design with black boxes) is affected.

6-2-3. Write out the remaining static-only checkpoint by executing the following command.

```
write_checkpoint -force Checkpoint/static_route_design.dcp
```

This static-only checkpoint would be used for any future configurations, but here, you simply keep this design open in memory.

Create Other Configurations Step 7

7-1. Read next set of RM dcps.

7-1-1. With the locked static design open in memory, read in post-synthesis checkpoints for the other two reconfigurable modules.

```
read_checkpoint -cell top_i/reconfig_addsub
Synth/rModule_addsub/subtractor/addsub_synth.dcp

read_checkpoint -cell top_i/reconfig_leds
Synth/rModule_leds/rightshift/shift_synth.dcp
```

7-1-2. Optimize, place and route the design by executing the following commands.

```
opt_design

place_design

route_design
```

7-1-3. Save the full design checkpoint by executing the following command.

```
write_checkpoint -force
Implement/Config_subtract_right/top_route_design.dcp
```

7-1-4. Save the checkpoints for each of the reconfigurable modules by issuing the following commands.

```
write_checkpoint -force -cell top_i/reconfig_addsub
Checkpoint/addsub_subtract_route_design.dcp

write_checkpoint -force -cell top_i/reconfig_leds
Checkpoint/shift_right_route_design.dcp
```

7-1-5. Execute the following command to close the project.

```
close_project
```

7-2. Create the blanking configuration.

7-2-1. Open the static route checkpoint.

```
open_checkpoint Checkpoint/static_route_design.dcp
```

7-2-2. For creating the blanking configuration, use the `update_design -buffer_ports` command to insert LUTs tied to constants to ensure the outputs of the reconfigurable partition are not left floating. Enter the following in the newly opened window.

```
update_design -buffer_ports -cell top_i/reconfig_addsub

update_design -buffer_ports -cell top_i/reconfig_leds
```

7-2-3. Now place and route the design. There is no need to optimize the design.

```
place_design
```

```
route_design
```

The base (or blanking) configuration bitstream, when it is generated in the next section, will have no logic for either reconfigurable partition. It will consist of outputs driven by ground. Outputs can be tied to VCC if desired, using the HD.PARTPIN_TIEOFF property.

7-2-4. Save the checkpoint in the `Config_blank` directory.

```
write_checkpoint -force Implement/Config_blank/top_route_design.dcp
```

7-2-5. Execute the following command to close the project.

```
close_project
```

Run PR_Verify Step 8

8-1. You must ensure that the static implementation, including interfaces to reconfigurable regions, is consistent across all Configurations. To verify this, you run the PR_Verify utility

8-1-1. Run the **pr_verify** command from the Tcl Console.

```
pr_verify -initial Implement/Config_add_left/top_route_design.dcp -
additional {Implement/Config_subtract_right/top_route_design.dcp
Implement/Config_blank/top_route_design.dcp}
```

When completed scroll through the Tcl Console window and notice the following:

```
DCP2: Implement/Config_subtract_right/top_route_design.dcp
  Number of reconfigurable modules compared = 2
  Number of partition pins compared         = 23
  Number of static tiles compared           = 224
  Number of static sites compared           = 231
  Number of static cells compared           = 686
  Number of static routed nodes compared    = 12083
  Number of static routed pips compared     = 11375
INFO: [Vivado 12-3253] PR_VERIFY: check points Implement/Config_add_left/top_route_design.dcp and Implement/Config_subtract_right/top_route_design.dcp are compatible
INFO: [Vivado 12-3501] pr verify Implement/Config add left/top route design.dcp Implement/Config blank/top route design.dcp

DCP2: Implement/Config_blank/top_route_design.dcp
  Number of reconfigurable modules compared = 2
  Number of partition pins compared         = 23
  Number of static tiles compared           = 224
  Number of static sites compared           = 231
  Number of static cells compared           = 686
  Number of static routed nodes compared    = 12083
  Number of static routed pips compared     = 11375
INFO: [Vivado 12-3253] PR_VERIFY: check points Implement/Config_add_left/top_route_design.dcp and Implement/Config_blank/top_route_design.dcp are compatible
```

Figure 15. PR_verify result

8-1-2. Execute the following command to close the project.

```
close_project
```

Generate Bit Files Step 9

9-1. After all the Configurations have been validated by PR_Verify, full and partial bit files must be generated for the entire project

9-1-1. Read the first configuration in the memory, using the command. in the original Vivado window

```
open_checkpoint Implement/Config_add_left/top_route_design.dcp
```

9-1-2. Generate the full and partial bitstreams for this design, making sure that the bit files are in a unique directory related to the full design checkpoint from which they were created. Run the following in the newly opened Vivado window:

```
write_bitstream -file Bitstreams/Config_addleft.bit

close_project
```

Notice the three bitstreams will be created.

Config_addleft.bit This is the power-up, full design bitstream.
Config_addleft_pblock_reconfig_addsub_partial.bit This is the partial bit file for the *adder* module.

Config_addleft_pblock_reconfig_leds_partial.bit This is the partial bit file for the *leftshift* module.

9-1-3. Generate full and partial bitstreams for the second configuration, again keeping the resulting bit files in an appropriate folder by executing the following commands.

```
open_checkpoint Implement/Config_subtract_right/top_route_design.dcp

write_bitstream -file Bitstreams/Config_subtractright.bit

close_project
```

The three bitstreams will be created with a different base name.

9-1-4. Generate a full bitstream with black boxes, plus blanking bitstreams for the reconfigurable modules. Blanking bitstreams can be used to “erase” an existing configuration to reduce power consumption. Before creating the bitstreams, use the “update_design -buffer_ports” command to insert LUTs tied to constants to ensure the outputs of the reconfigurable partition are not left floating.

```
open_checkpoint Implement/Config_blank/top_route_design.dcp

write_bitstream -file Bitstreams/blanking.bit

close_project
```

The base configuration bitstream will have no logic for either reconfigurable partition, simply outputs driven by ground. Outputs can be tied to VCC if desired, using the HD.PARTPIN_TIEOFF property.

Test the Design — Step 10

10-1. Connect the board with one micro-USB cable (PROG UART connector). Place the board in the SD boot mode. Copy the provided BOOT.bin file on the SD Card and place the SD card in the board. Power On the board. Establish the connection with the Hardware Manager and program the chip with the blanking.bit bitstream file.

10-1-1. Make sure that one micro-usb cable is connected between the PC and the PROG UART connector of the board.

10-1-2. Make sure that the board is set to boot in SD card boot mode (left most two pins).

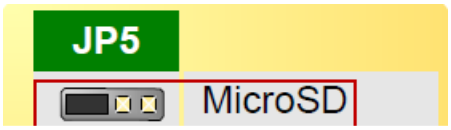


Figure 16: Setting up to boot off the Micro-SD card

10-1-3. Using the Windows Explorer, copy the **BOOT.bin** from the *c:/xup/PR/uart_led_lab/Sources/BOOT* directory into the root directory of a SD Card.

10-1-4. Place the SD Card in the board and power ON the board.

10-1-5. Start the Vivado Hardware Manager by clicking on the **Open Hardware Manager** under the *Tasks* section, or selecting from the *Tools* menu.

10-1-6. Select **Open Target > Auto Connect**

10-1-7. Right-click on **xc7z010_1** and select **Program Device…**

10-1-8. Click on the Browse button, browse to **c:/xup/PR/labs/uart_led_lab/Bitstreams**, select **blanking.bit**, and click **OK**.

10-1-9. Click **Program.**

You should see the bitstream is downloaded, the DONE LED turned ON, and the 4 LEDs are OFF, since there is no active *shift* RM in the blanking.bit file.

10-2. Start a terminal emulator program such as TeraTerm or HyperTerminal. Select an appropriate COM port (you can find the correct COM number using the Control Panel). Set the COM port for 115200 baud rate communication.

10-2-1. Start a terminal emulator program such as TeraTerm or HyperTerminal.

10-2-2. Select the appropriate COM port (you can find the correct COM number using the Control Panel).

10-2-3. Set the *COM* port for **115200** baud rate communication.

10-3. Start a SDK session, point it to the c:/xup/PR/labs/uart_led_lab/Sources/uart_through_gpio.sdk workspace.

10-3-1. Open **SDK** by selecting **Start > All Programs > Xilinx Design Tools > Vivado 2014.3 > Xilinx SDK 2014.3**

10-3-2. In the **Select a workspace** window, click on the browse button, browse to the *c:\xup\PR\labs\uart_led_lab\Sources* directory, select *uart_through_gpio.sdk,* and click **OK.**

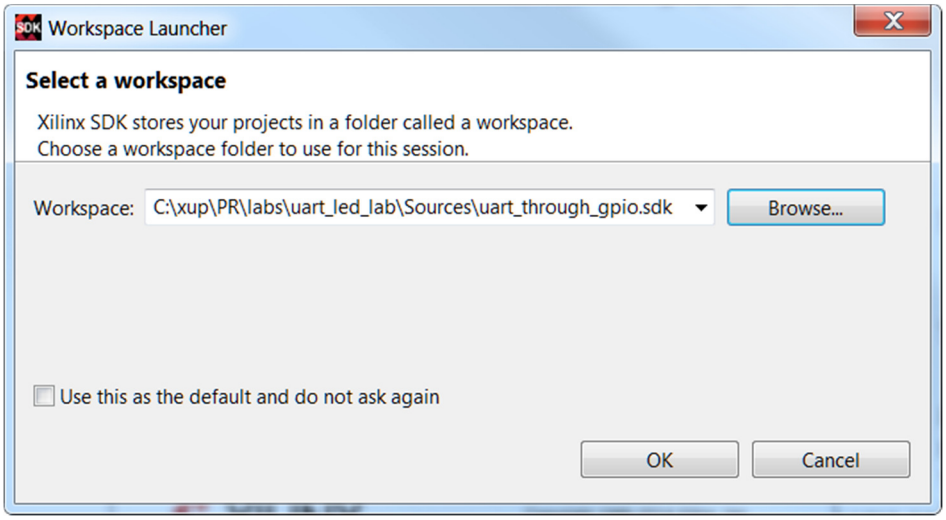


Figure 17. Setting up SDK workspace

10-3-3. Click **OK**.

10-3-4. In the *Project Explorer*, right-click on the *TestApp*, select *Run As*, and then **Launch on Hardware (GDB).**

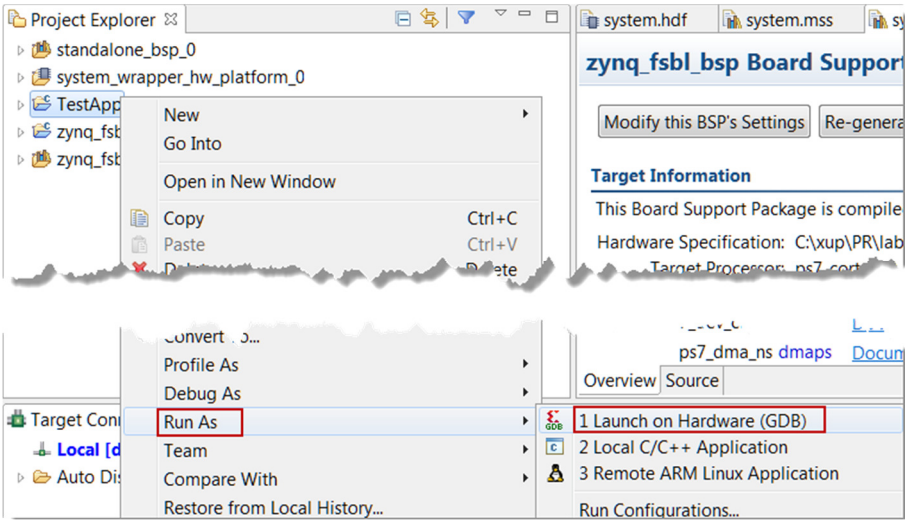


Figure 18. Running the application

The program will be downloaded and the execution will begin indicated by the Terminate button ( ) in SDK.

10-3-5. In the serial terminal window, type **?** followed by the *Enter* key.

You should see **+** as the response.

10-4. Program the device with various partial bitstreams and verify the functionality by entering various operands and pressing BTN0, BTN1, and BTN2 on the board. Note the time it takes to program each RP compared to the entire design bitstream.

10-4-1. In Vivado, right-click on **xc7z010_1** and select **Program Device…**

10-4-2. Click on the Browse button, browse to **c:/xup/PR/labs/uart_led_lab/Bitstreams**, select **Config_addleft_pblock_reconfig_addsub_partial.bit**, and click **OK**.

10-4-3. Click **Program.**

Notice how quickly the device gets programmed. The adder module is loaded.

10-4-4. In the serial terminal window, type **?** followed by the *Enter* key.

You should see **+** as the response since now the adder module is loaded.

10-4-5. In the serial terminal window, type the following commands (pressing Enter key after each line) to verify the adder functionality.

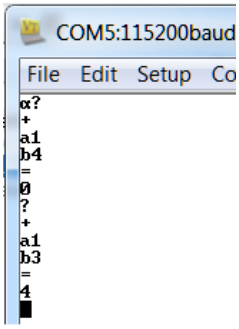


Figure 19. Terminal window showing add operation

10-4-6. Similarly, program the device with **Config_addleft_pblock_reconfig_leds_partial.bit** and notice that only the least significant LED is ON.

10-4-7. Press **BTN0** and observe the LEDS starts shifting left.

10-4-8. Similarly, program the device with **Config_subtractright_pblock_reconfig_addsub_partial.bit** file. Enter ? in the terminal and observe that **–** is displayed. Enter the following commands and observe the subtract operation.

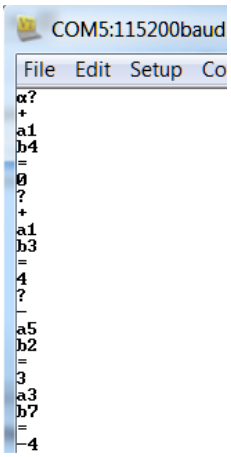


Figure 20. Terminal window showing subtract operation

10-4-9. Similarly, program the device with **Config_subtractright_pblock_reconfig_leds_partial.bit** and notice that LEDS are shifting right.

You can press **BTN1** to stop the shifting; **BTN0** to restart the shifting, or **BTN2** to reset the shifter (since the right shifting module is loaded) and it will show LED3 ON.

10-4-10. When satisfied, close the hardware manager by selecting **File > Close Hardware Manager**.

10-4-11. Click on the *Terminate* button in the SDK.

10-4-12. Close the SDK by selecting **File > Exit**.

10-4-13. Close Vivado by selecting **File > Exit**.

10-4-14. Power OFF the board.

Conclusion

This lab showed you steps involved in generating partial reconfiguration design using Vivado 2014.3 with partial reconfiguration feature enabled. You used the provided Tcl scripts to generate the static design that consists of the processing system to configure the UART port as a simple GPIO port and extend the signals to the programmable logic sub-system where they are connected to the actual design. You used another script to synthesize the RMs in a bottom-up approach which is required for the PR flow. You create the configurations and implemented the design. You generated the full and partial bitstreams, downloaded them and verified the functionality using the development board.