Using PRC for Hardware Trigger and Debug Lab

Introduction

In this lab, you will use the Partial Reconfiguration Controller (PRC) core to reconfigure a design that has one RP having two RMs. The provided PRC core is a Beta version which will be available in Vivado 2015.1 release. You will use the integrated logic analyzer (ILA) core to monitor the ICAP signals. You will go through the design process and then use the provided design checkpoint to implement the design. You will continue 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 having a PS7 sub-system
- Add PRC in the system
- Add an Integrated Logic Analyzer core to monitor ICAP ports
- Configure the PRC to enable partial reconfiguration using hardware triggers generated by pushbuttons
- Use the provided static dcp (design checkpoint) having the PRC functionality
- Use various Tcl scripts to synthesize the RMs, floorplan the design, add the RMs, create multiple configurations, implement the design and generate the full and partial bitstreams for various configurations
- Use Xilinx SDK program to create an application and a bootable BOOT.bin file
- Copy the generated bitstreams and the BOOT.bin on a SD Card and verify partial reconfigurable design functionality

Design Description

The purpose of this lab exercise is to implement a design that is dynamically reconfigurable using the PRC when a hardware event occurs. The hardware events are generated by pressing on-board pushbuttons. The design, shown in Figure 1, consists of the PRC and one RP. The RP has two functional RMs performing right and left shifting pattern on LEDs. The dynamic partial reconfigurable modules are updated using the hardware triggers.

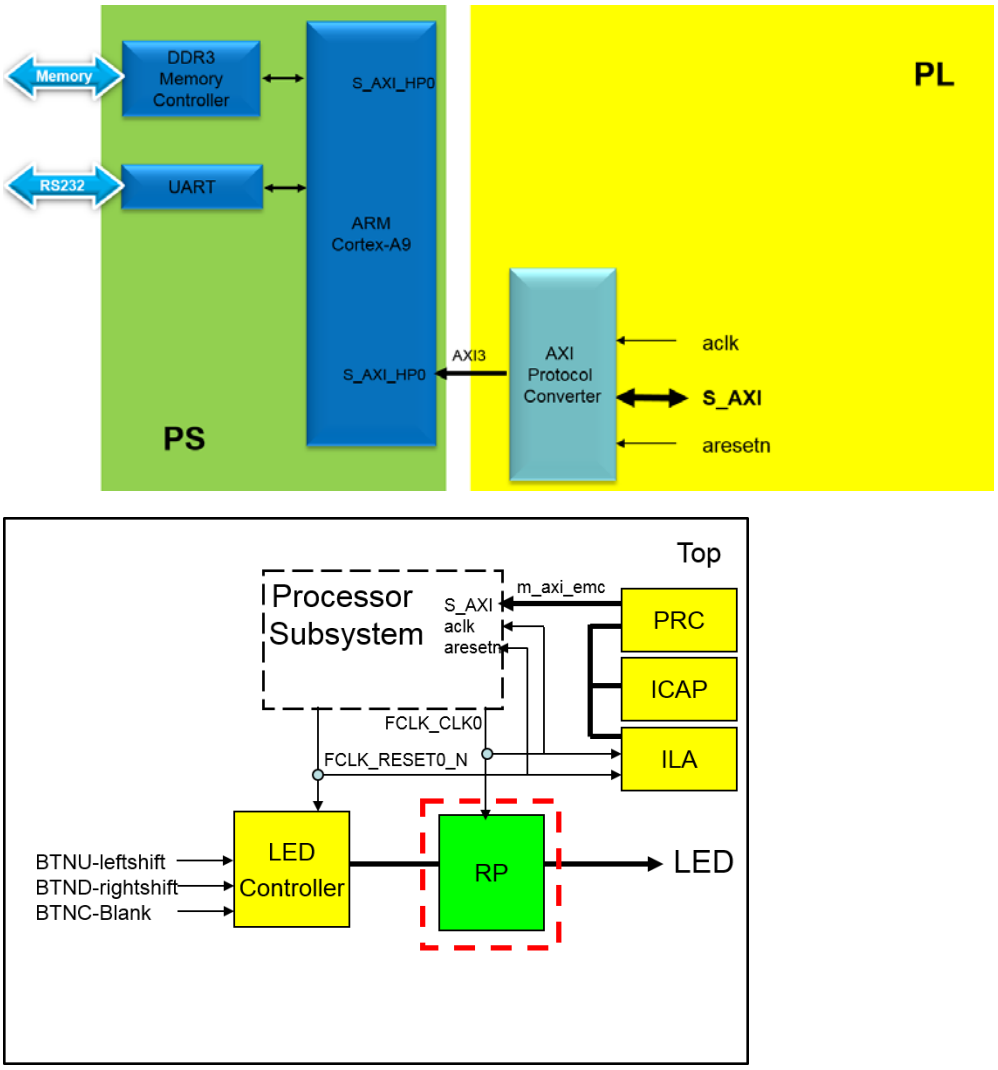
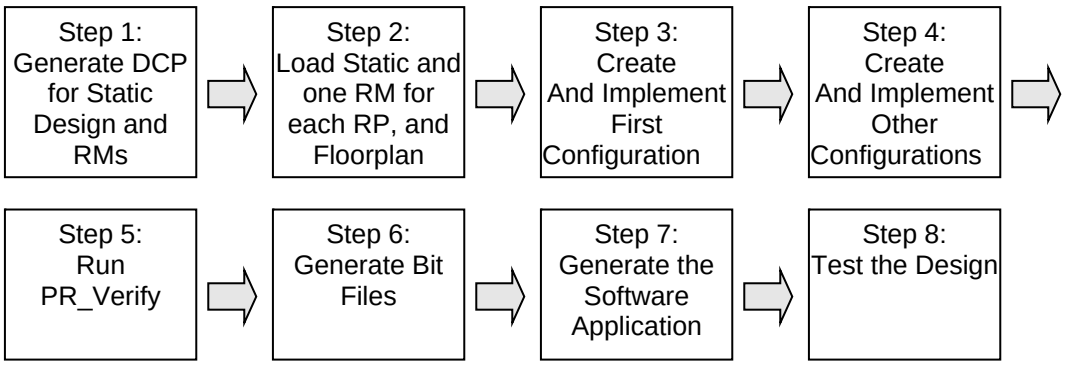


Figure 1. A Complete System

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 RMs Step 1

1-1. Start Vivado and execute the provided Tcl script to create the design having one RP.

1-1-1. Open **Vivado** by selecting **Start > All Programs > Xilinx Design Tools > Vivado 2016.3 > Vivado 2016.3**

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/prc_hw_trigger_ila_lab
```

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

`source ps7_create_zed.tcl` (for ZedBoard) or

`source ps7_create_zybo.tcl` (for Zybo)

This script will create a block design called *system*. It will:

- Instantiate ZYNQ PS with *SD 0* and *UART 1* peripherals enabled, *S_HP0* (in 32-bit mode) interface enabled, and *FCLK_CLK0* and *FCLK_RESET0_N* ports enabled
- Add an *axi_protocol_converter* instance and configure it (Figure 2) to AXI4 protocol on the slave side and AXI3 protocol on the master side. The READ_WRITE_MODE is set to READ_ONLY since the PRC will only read the partial bitstreams from the DDR memory. Connect the added *axi_protocol_converter* instance's **M_AXI** interface to the **S_HP0** interface of the processor system. Make the **S_AXI**, **aclk**, and **aresetn** ports external

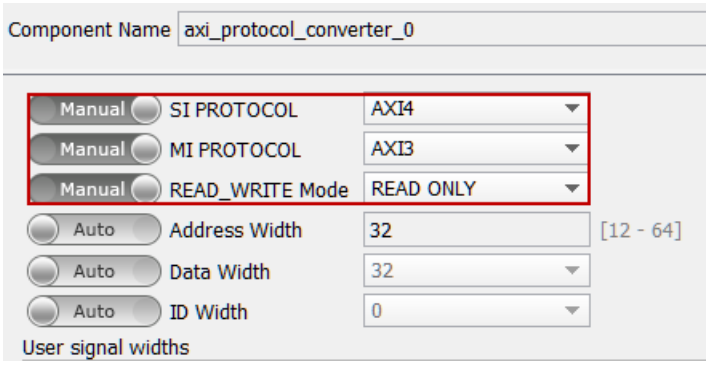


Figure 2. Configuring AXI Protocol Converter IP

- Make **FCLK_CLK0** and **FCLK_RESET0_N** external
- Connect **aclk** net to the **S_AXI_HP0_ACLK** port
- The design looks like as shown in Figure 3.

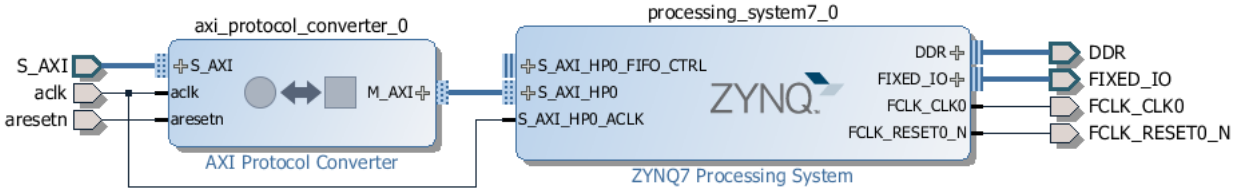


Figure 3. The processor system

- The drc will be run next to make sure that there are no design violations, the wrapper file will be created, and the block design will be generated.
- Once the wrapper file is generated, the script will add the provided *top.v* and rest of the modules for the static design. The design hierarchy will look like as shown in Figure 4.

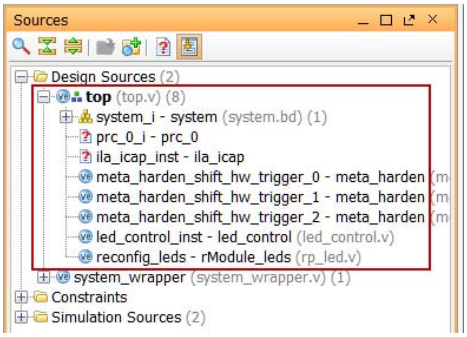


Figure 4. The design hierarchy including the processor system and other top-level modules

In the sources tab, notice the *prc_0* and *ila_0* instances have "?" since those modules have not been added yet.

1-2. Add an instance of the PRC core. Set the global options.

1-2-1. Click on the *IP Catalog* in the Flow Navigator pane.

1-2-2. Expand the sub-catalog and notice the *Partial Reconfiguration Controller* IP entry under the **Partial Reconfiguration** sub-folder.

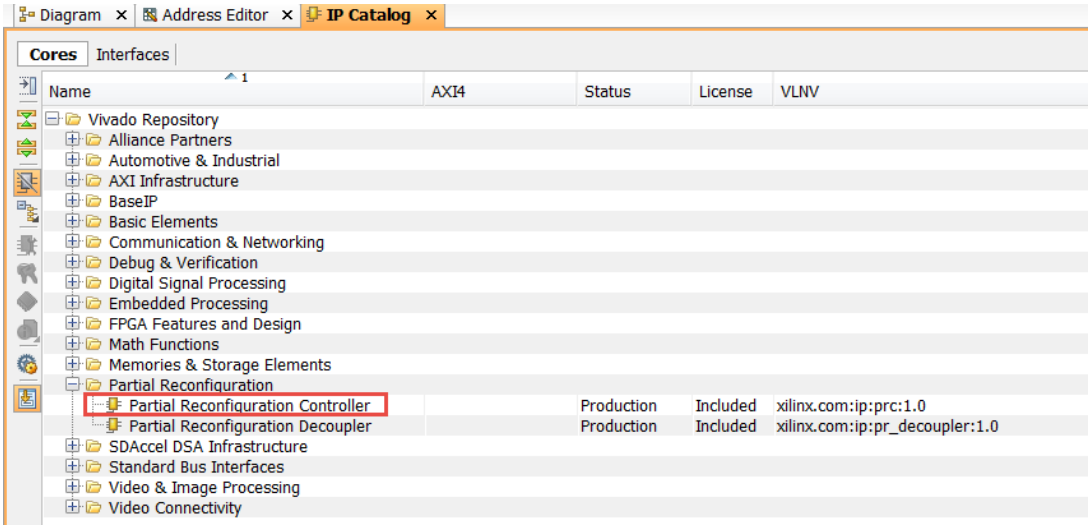


Figure 5. The IP Catalog

1-2-3. Double-click on the **Partial Reconfiguration Controller** entry and click on the **Customize IP** button to open the *Customize IP* form.

1-2-4. Change the *number of clock domain crossing stages* to **2** since we have a single clock domain design.

1-2-5. Uncheck the Enable the AXI Lite interface option as we will not be using that interface.

1-2-6. Change the *Specify the CAP arbitration protocol* option to **1) Latency has not been added to arbiter signals**

1-2-7. Leave rest of the global options to default values. Note that the reset signal is active_low (FCLK_RESET0_N is connected to the port).

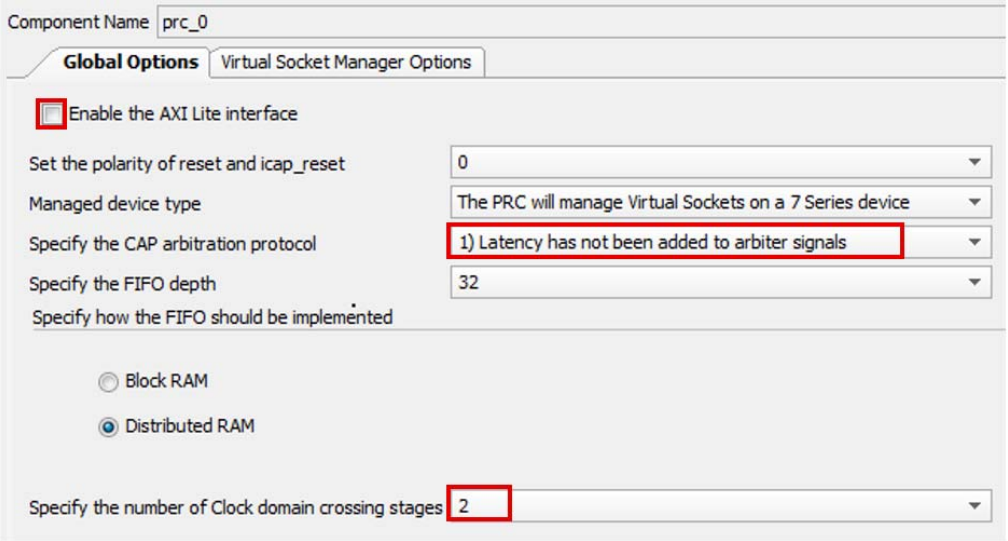


Figure 6. Setting the global options

1-3. Create a new virtual socket manager naming it as `rp_led`

1-3-1. Click on the **Virtual Socket Manager Options** tab.

1-3-2. Enter **rp_led** in the *Virtual Socket Name* field and change the Number of RMs allocated from 2 to 3.

Notice the **rp_led** is moved to the *Virtual Socket Manager to configure* field and the *Number of RMs allocated* changed from 3 to 4 since it requires power of 2 values.

1-3-3. Change the *Number of Hardware Triggers* to **3** since we will have two RMs and one blanking RM.

1-3-4. Change the *Number of Triggers allocated* to **3** and notice that the number changes to power of 2 when you click in some other field.

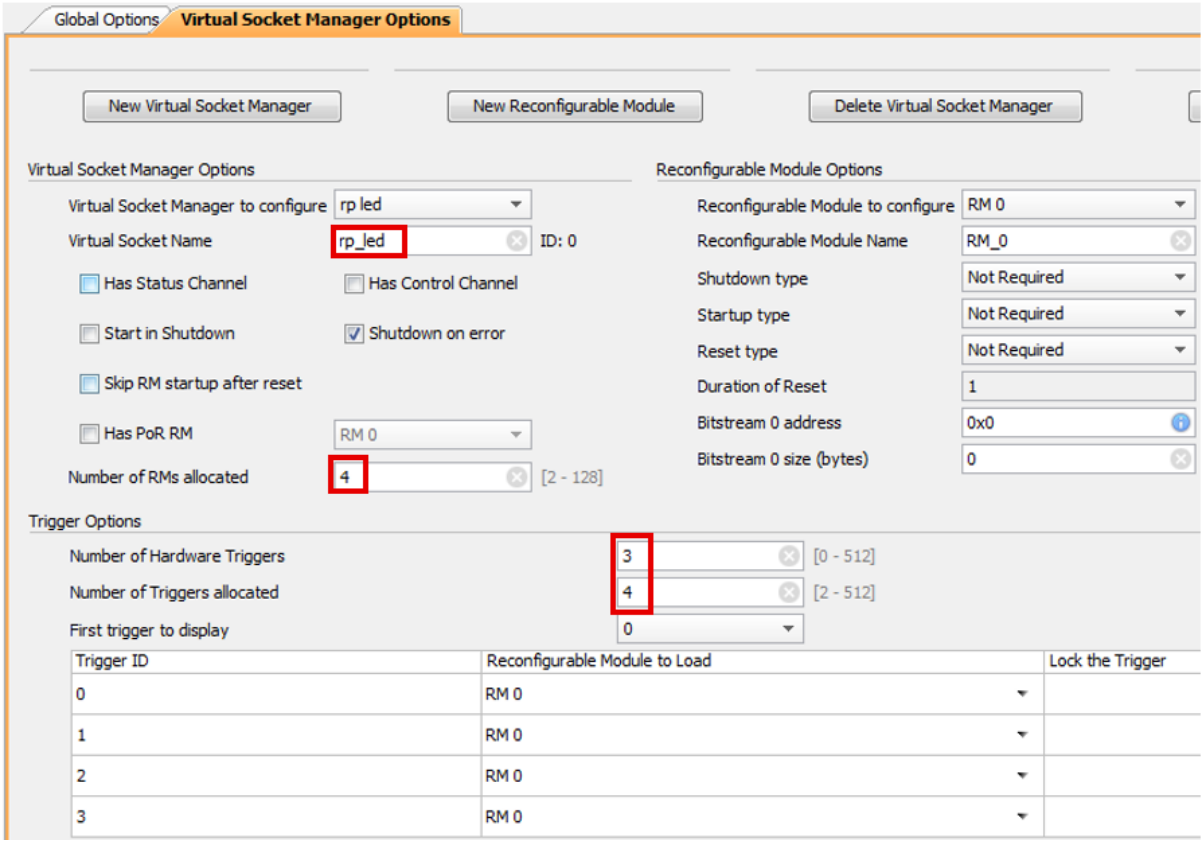


Figure 7. Creating the Virtual Socket Manager

1-4. Create three reconfigurable modules and customize them with the bitstream addresses and sizes

1-4-1. Under the Reconfigurable Module Options, enter **left** in the *Reconfigurable Module Name* field.

1-4-2. Set the Bitstream address to **0x00200000** and the size to **151484 (for ZedBoard) or 106236 (for Zybo).**

Notice the name has moved to Reconfigurable Module to *Configure* field.

The bitstream size is pre-determined based on the RP floorplan and the resources used. If this information is not available then you will have to go through one additional pass to generate the partial bitstreams, determine the size, then make changes to the core customization, and then regenerate the bitstreams.

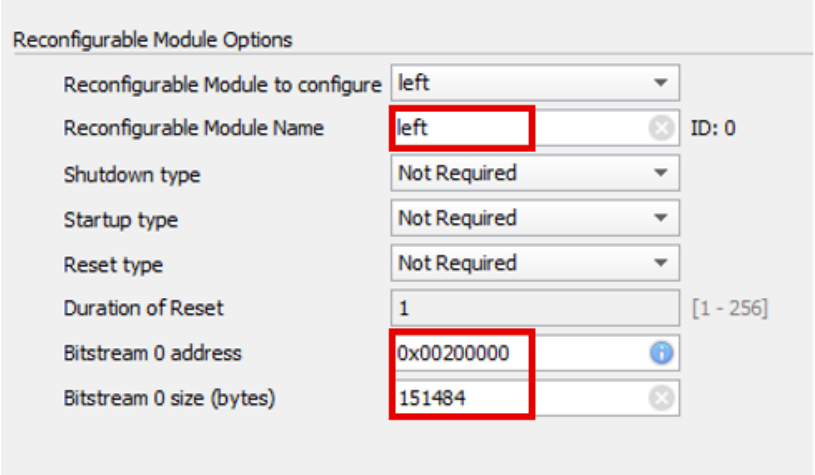


Figure 8. Setting the bitstream address and the size for the RM

1-4-3. Similarly, click on **New Reconfigurable Module** and create two more RMs- one called **right** and another called **blank.**

1-4-4. Set the bitstream address to **0x00300000** and *size* to **151484 (for ZedBoard) or 106236 (for Zybo)** for the *right* and **0x00400000** for the *blank* and size to **121960 (for ZedBoard) or 76712 (for Zybo)**.

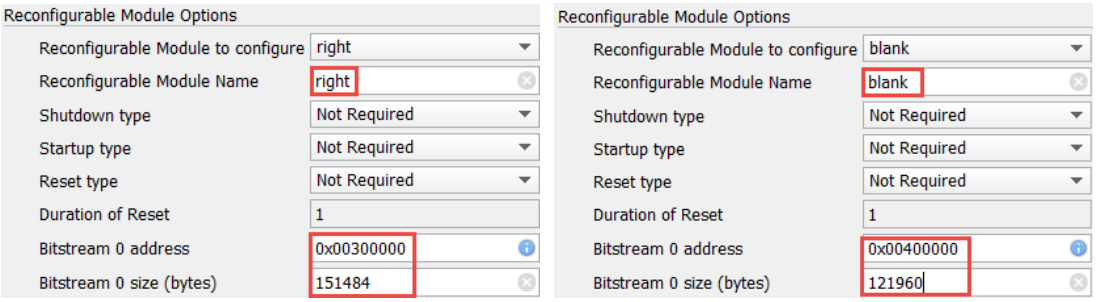


Figure 9. Settings of the other two RMs

1-4-5. Notice the Trigger Options assigns Trigger ID *0* to **left**, *1* to **right**, and *2* to **blank**.

Using the drop-down button one can change the ID assignments.

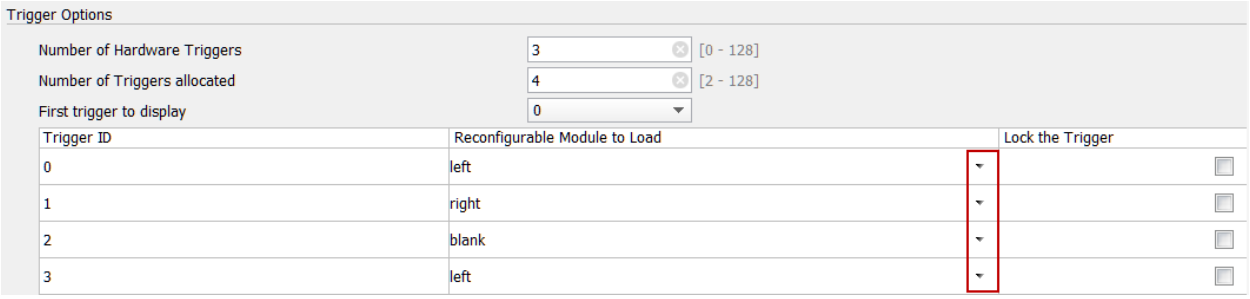


Figure 10. The Trigger Options

1-4-6. Click **OK** to accept the settings and add the IP to the project.

1-4-7. In the **Generate Output Products** form, select the *global* option, then click on **Apply**, and then click on **Skip** to close the synthesis option form.

1-5. Add an ILA instance with the necessary number and appropriate size ports so all ICAP ports can be monitored.

1-5-1. In the IP Catalog expand **Debug & Verification > Debug** and double-click on the *ILA* entry and click on the **Customize IP** button to open the *Customize IP* form.

1-5-2. Change the instance name to **ila_icap** and double-click to open the customization window.

1-5-3. Select *Native* as a **Monitor Type** and set the *Number of Probes* to **4**.

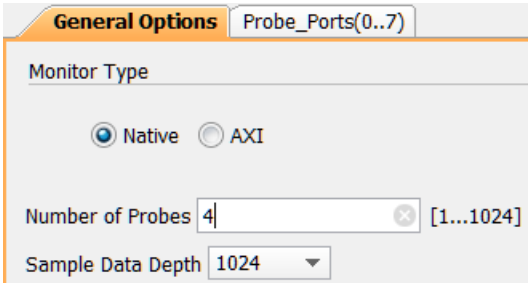


Figure 11. ILA customization- Monitor Type and setting number of probes

1-5-4. Click on the *Probes_Ports(0..7)* tab and change the size to **32, 32, 1, 1**, and click **OK**.

We have 32-bit input, 32-bit output, cs, and rdwr ports on the ICAP and we want to monitor those ports.

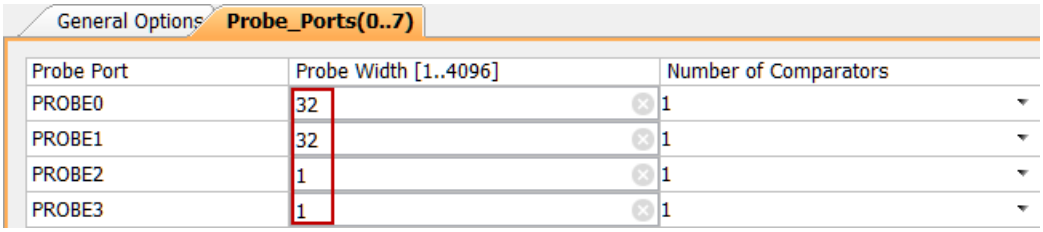


Figure 12. Setting the probes widths

1-5-5. In the **Generate Output Products** form, select the *global* option, then click on **Apply**, and then click on **Skip** to close the synthesis option form.

1-5-6. Click on **Run Synthesis** to synthesize the static design and generate the dcp.

1-5-7. When the synthesis process is completed, click **Cancel**.

1-5-8. Close the project.

1-5-9. Using the Windows Explorer, copy the **top.dcp** file from c:\xup\PR\labs \prc_hw_trigger_ila_lab\prc_hw_trigger_ila_<board>lab\prc_hw_trigger_ila_<board>_lab.runs\syn

th_1 into the *Synth**Static* directory under the current lab directory. Substitute zed or zybo for <board>.

1-5-10. Copy design checkpoints for the axi_protocol_converter_0 and processing_system7_0 instances. These .dcp files are also in their respective folders under the synth_1 folder of the prc_hw_trigger_ila_<board>_lab.runs directory. Move system_axi_protocol_converter_0_0.dcp and system_processing_system7_0_0.dcp to *Synth**Static* to sit alongside top.dcp.

1-6. Generate the dcp for each of the RMs.

1-6-1. Make sure that you are in the lab directory otherwise enter the following command to change to the lab directory and hit **Enter**.

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

1-6-2. Synthesize each of the RMs (two) executing the provided Tcl script.

`source synth_reconfig_modules_zed.tcl` (for ZedBoard) or

`source synth_reconfig_modules_zybo.tcl` (for Zybo)

This script will add the HDL files for a given RM, synthesize the module(s) for the RM 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.

Load Static and one RM for each RPs, and Floorplan Step 2

2-1. Load the static and one RM design for each of the RPs.

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

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

2-1-2. Execute the following Tcl script to load the design checkpoint and floorplan the reconfigurable region.

`source floorplan_design_zed.tcl` (for ZedBoard) or

`source floorplan_design_zybo.tcl` (for Zybo)

The script will do the following:

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

  ```
  open_checkpoint Synth/Static/top.dcp
  ```

- Load the IP checkpoint for the AXI Protocol Converter by using the **read_checkpoint** command.

  ```
  read_checkpoint -cell system_i/axi_protocol_converter_0
  Synth/Static/system_axi_protocol_converter_0_0.dcp
  ```

- Load the IP checkpoint for the Processing System by using the **read_checkpoint** command.

```
read_checkpoint -cell system_i/processing_system7_0
Synth/Static/system_processing_system7_0_0.dcp
```

- Load one RM for the RP by using the **read_checkpoint** command.

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

- 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 reconfig_leds]
  ```

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

  ```
  write_checkpoint –force Checkpoint/top_link_left.dcp
  ```

- Read the provided floorplan constraints file which defines the RP regions.

  `read_xdc Sources/xdc/fplan_zed.xdc` (for ZedBoard) or

  `read_xdc Sources/xdc/fplan_zybo.xdc` (for Zybo)

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

  `read_xdc Sources/xdc/top_io_zed.xdc` (for ZedBoard) or

  `read_xdc Sources/xdc/top_io_zybo.xdc` (for Zybo)

Create and Implement the First Configuration Step 3

3-1. Create and implement the first configuration.

3-1-1. Execute the following command from the Tcl console after making sure that the working directory is set to `cd c:/xup/PR/labs/prc_hw_trigger_ila_lab`.

```
source create_first_configuration.tcl
```

The script will do the following:

- Optimize, place and route the design by executing the following commands:

  ```
  opt_design

  place_design

  route_design
  ```

- Save the full design checkpoint and create report files by executing the following commands:

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

  report_utilization -file
  Implement/Config_left/top_utilization.rpt
  ```

- Save checkpoints for each of the reconfigurable modules by issuing these two commands:

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

- Write the debug_netx.ltx file which will be used in the Vivado Hardware Analyzer in the Testing step.

```
write_debug_probes –force
./Implement/Config_left/debug_nets.ltx
```

3-2. 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.

3-2-1. Execute the following command to update the design with the blackbox and write the checkpoint.

```
source lock_placement_with_blackbox.tcl
```

The script will do the following tasks:

- Clear out the existing RMs executing the following commands.

```
update_design -cells reconfig_leds -black_box
```

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

```
lock_design -level routing
```

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

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

Create and Implement Other Configurations Step 4

4-1. Read next set of RM dcps, create and implement the second configuration.

4-1-1. Execute the following command to create and implement the second configuration

```
source create_second_configuration.tcl
```

The script will do the following tasks:

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

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

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

```
opt_design

place_design

route_design
```

- Save the full design checkpoint by executing the following command.

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

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

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

- Close the project

```
Close_project
```

4-2. Create the blanking configuration.

4-2-1. Execute the following command to create and implement the second configuration

```
source create_blanking_configuration.tcl
```

The script will do the following tasks:

- Open the static route checkpoint.

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

- 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.

```
update_design -buffer_ports -cell reconfig_leds
```

- 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 we generate in the next section, 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.

- Save the checkpoint in the `Config_blank` directory.

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

- Close the project

```
Close_project
```

Run PR_Verify Step 5

5-1. 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

5-1-1. Run the **pr_verify** command from the Tcl Console.

```
source verify_configurations.tcl
```

The script will perform the following tasks:

- execute the pr_verify command and then close the project:

```
pr_verify -initial Implement/Config_left/top_route_design.dcp -
additional {Implement/Config_right/top_route_design.dcp
Implement/Config_blank/top_route_design.dcp}
```

You should see the message indicating the `Config_left` configuration is compatible with `Config_right`, and the `Config_left` configuration is compatible with `Config_blank`.

- Execute the following command to close the project.

```
close_project
```

Generate Bit Files Step 6

6-1. After all the Configurations have been validated by PR_Verify, full and partial bit files must be generated for the entire project

6-1-1. Generate the full configurations and partial bitstreams by executing the following tcl script.

```
source generate_bitstreams.tcl
```

6-1-2. The script will do the following tasks:

- Read the first configuration in the memory and generate the bitstreams both in *bit* and *bin* formats

```
open_checkpoint Implement/Config_left/top_route_design.dcp

write_bitstream -file Bitstreams/Config_left.bit -force

write_cfgmem -format BIN -interface SMAPx32 -disablebitswap
-loadbit "up 0
Bitstreams/Config_left_pblock_reconfig_leds_partial.bit"
Bitstreams/left.bin

close_project
```

- Generate the bitstreams for the second configuration

```
open_checkpoint Implement/Config_right/top_route_design.dcp

write_bitstream -file Bitstreams/Config_right.bit -force
```

```
write_cfgmem -format BIN -interface SMAPx32 -disablebitswap
-loadbit "up 0
Bitstreams/Config_right_pblock_reconfig_leds_partial.bit"
Bitstreams/right.bin

close_project
```

- Generate the bitstreams with black boxes.

```
open_checkpoint Checkpoint/static_route_design.dcp

write_bitstream -file Bitstreams/blanking.bit -force

write_cfgmem -format BIN -interface SMAPx32 -disablebitswap
-loadbit "up 0
Bitstreams/blanking_pblock_reconfig_leds_partial.bit"
Bitstreams/b_led.bin

close_project
```

Generate the Software Application Step 7

7-1. Open the PS design that was created in Step 1. Export the hardware design and launch SDK.

7-1-1. In Vivado, click on the **Open Project** link, browse to *c:/xup/PR/labs/prc_hw_trigger_ila_lab/prc_hw_trigger_ila_<board>_lab*, select the *prc_hw_trigger_ila_<board>_lab.xpr* and click **OK** to open the design created in Step 1.

7-1-2. Select **File > Export > Export Hardware…**

7-1-3. In the *Export Hardware* form, make sure that the *Include bitstream* checkbox is not checked and click **OK**.

7-1-4. Select **File > Launch SDK**

7-1-5. Click **OK** to launch SDK.

The SDK program will open. Close the Welcome tab if it opens.

7-2. Create a Board Support Package enabling FAT file system.

7-2-1. In **SDK**, select **File** > **New** > **Board Support Package.**

7-2-2. Click **Finish** with the default settings (with standalone operating system).

This will open the Software Platform Settings form showing the OS and libraries selections.

7-2-3. Select **xilffs** as the FAT file support is necessary to read the partial bit files.

7-2-4. Click **OK** to accept the settings and create the BSP.

7-3. Create an application.

7-3-1. Select **File** > **New** > **Application Project.**

7-3-2. Enter **TestApp** as the *Project Name*, and for *Board Support Package*, choose **Use Existing** (*standalone_bsp_0* should be the only option).

7-3-3. Click **Next**, and select *Empty Application* and click **Finish.**

7-3-4. Expand the **TestApp** entry in the project view, right-click the *src* folder, and select **Import.**

7-3-5. Expand **General** category and double-click on **File System.**

7-3-6. Browse to *c:\xup\PR\labs\prc_hw_trigger_ila_lab\Sources\TestApp\src* and click **OK.**

7-3-7. Select **TestApp.c** and click **Finish** to add the file to the project.

7-3-8. Right-click on TestApp and select **C/C++ Building Settings.**

7-3-9. Select **ARM v7 GCC Compiler > Symbols,** and click **+**

7-3-10. Enter **ZED** for ZedBoard or **ZYBO** for Zybo, and click **OK.**

The program should compile successfully.

Open the source file and verify that the bin file size in the program listed matches the size you made a note earlier (except it is 4x as the program uses the size in words). If different, then change in the program and save it.

7-4. Create a zynq_fsbl application.

7-4-1. Select **File** > **New** > **Application Project.**

7-4-2. Enter **zynq_fsbl** as the *Project Name*, and for *Board Support Package*, choose **Create New**.

7-4-3. Click **Next**, select *Zynq FSBL,* and click **Finish.**

This will create the first stage bootloader application called zynq_fsbl.elf

7-5. Create a Zynq boot image.

7-5-1. Select **Xilinx Tools** > **Create Boot Image.**

7-5-2. Click the Browse button of the Output BIF file path field, browse to *c:\xup\PR\labs\prc_hw_trigger_ila_lab,* and then click **Save** with the *output* as the default filename.

7-5-3. Click on the **Add** button of the *Boot image partitions,* click the Browse button in the Add Partition form, browse to

c:\xup\PR\labs\prc_hw_trigger_ila_lab\prc_hw_trigger_ila_<board>_lab\prc_hw_trigger_ila_<board>_lab.sdk\zynq_fsbl\Debug directory, select *zynq_fsbl.elf* and click **Open**.

7-5-4. Click **OK**.

7-5-1. Click again on the **Add** button of the *Boot Image partitions,* click the Browse button in the Add Partition form, browse to **c:\xup\PR\labs\prc_hw_trigger_ila_lab\Bitstreams** directory, select *blanking.bit* and click **Open**.

7-5-2. Click **OK**.

7-5-3. Click again on the **Add** button of the *Boot Image partitions,* click the Browse button in the Add Partition form, browse to **c:\xup\PR\labs\prc_hw_trigger_ila_lab\prc_hw_trigger_ila_<board>_lab\prc_hw_trigger_ila_<board>_lab.sdk\TestApp\Debug** directory, select *TestApp.elf* and click **Open**.

7-5-4. Click **OK**.

7-5-5. Make sure that the output path is **c:\xup\PR\labs\prc_hw_trigger_ila_lab** and the filename is *BOOT.bin,* and click **Create Image.**

7-5-6. Close the SDK program by selecting **File > Exit**.

Test the Design Step 8

8-1. Place the board in the SD boot mode. Copy the BOOT.bin file on the SD Card. Copy the partial bin files generated in the bitstreams directory on the SD card, and place the SD card in the board. Power On the board.

8-1-1. Make sure that the board is set to boot in SD card boot mode.

8-1-2. Using the Windows Explorer, copy the **BOOT.bin** from the *c:/xup/PR/prc_hw_trigger_ila_lab/* directory on to a SD Card.

8-1-3. Copy the three partial bin files from the `bitstreams` directory.

8-1-4. Place the SD Card in the board and power ON the board.

You should see the four leds OFF as the design is loaded with blank RP.

At this point you can press **BTNU** (ZedBoard) or **BTN1** (Zybo) to reconfigure with the left shift functionality and **BTND** (ZedBoard) or **BTN2** (Zybo) with right shift. When you press **BTNC** (ZedBoard) or **BTN0** (Zybo) it will reconfigure the RP with blanking bitstream.

8-1-5. When satisfied, power OFF the board.

8-1. Connect the board with one micro-USB cable to JTAG port. Corrupt the left.bin to have corrupted sync word and b_led.bin with the corrupted

idcode. Copy them on the SD card overwriting the clean bin files. Place the SD card in the board. Power On the board.

8-1-1. Make sure that one micro-usb cable is connected between the JTAG port of the board and the PC.

8-1-2. Using a Hex Editor, open **left.bin** file and make change to the SYNC word so it looks like as shown below and then save the file as **sync.bin**

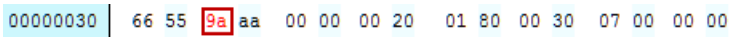


Figure 13. Corrupting the sync word

8-1-3. Similarly, open the **b_led.bin** file, change the IDCODE field as shown below, and save it as **idcode.bin**

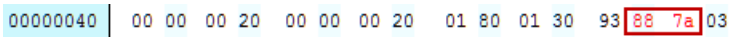


Figure 14. Corrupting the id code

8-1-4. Close the Hex editor program.

If you don't have an access to the hex editor or equivalent, copy the **BOOT.bin. sync.bin, right.bin** and **idcode.bin** files from the *bitstreams_debug* folder and place them on the SD card.

8-1-5. Rename the **sync.bin** to **left.bin** and **idcode.bin** to **b_led.bin** on the SD card. The renaming is necessary as the software application is looking for the named files.

8-1-6. Insert the SD card into the board, and power ON the board.

8-1-7. In Vivado, make sure that the working directory in the Tcl shell is `c:/xup/PR/labs/prc_hw_trigger_ila_lab`. If not then set it using the **cd** command.

8-1-8. Enter the following command to open the hardware manager, program the FPGA, and open the hardware manager's dashboard.

```
source run_ila_zed.tcl (for ZedBoard)

source run_ila_zybo.tcl (for Zybo)
```

8-1-9. Click the *Stop Trigger* button (■) to see the waveform.

8-1-10. Select **Window > Debug Probes**

8-1-11. Select **icap_prim_csib** probe in the *Debug Probes* window, right-click, and select **Add Probes to Basic Trigger Setup**

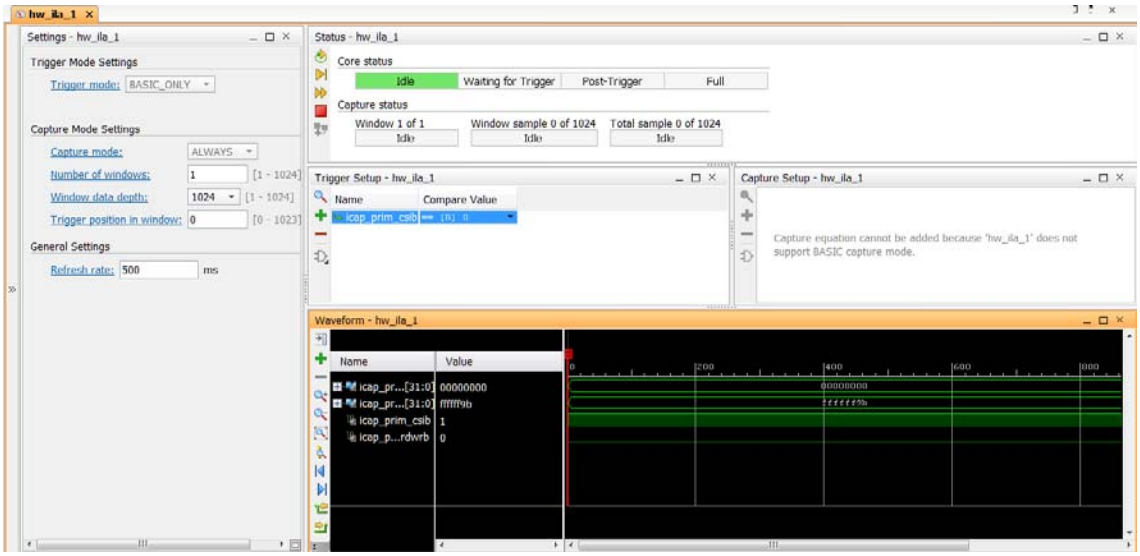


Figure 15. ILA dashboard

8-1-12. Click the *Run Trigger* button ( ) to trigger the ILA.

You will see that the run is waiting for the trigger condition to occur, which is writing to the ICAP.

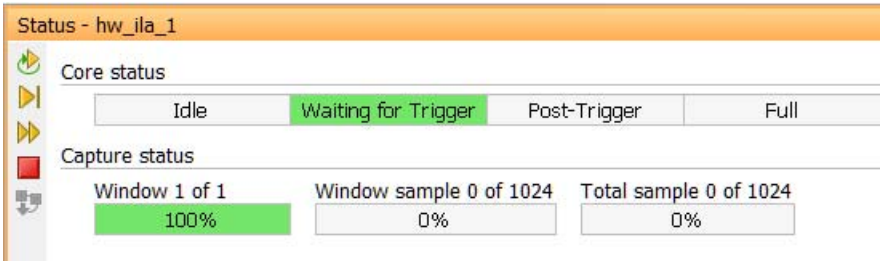


Figure 16. Waiting for the trigger condition to occur

8-1-13. Press the **BTND** (ZedBoard) or **BTN2** (Zybo) and observe the LEDs are shifting right and the ILA has triggered. Zoom in into the beginning part of the waveform and notice the SYNC word is detected and the ICAP output changes at sample *14*.

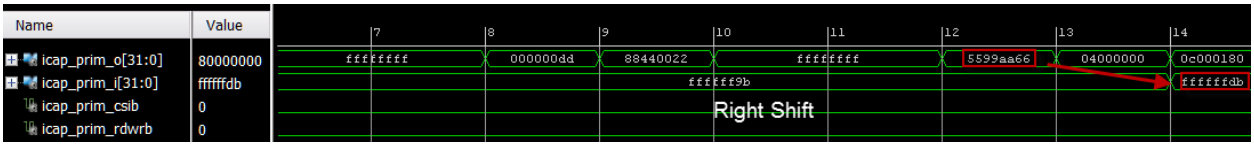


Figure 17. SYNC word detected

8-1-14. Click the *Run Trigger* button and then switch to the ILA-hw_ila_1 tab and observe that it is waiting for the trigger to occur.

8-1-15. Press the **BTNU**(ZedBoard) or **BTN1** (Zybo) to trigger the ILA and then observe the waveform.

Notice that the LEDs are still shifting right, however at 14 the ICAP output did not change.

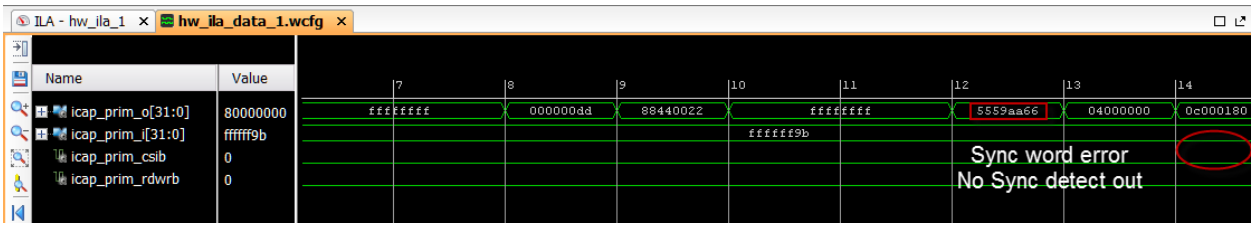


Figure 18. SYNC word error

8-1-16. Click the *Run Trigger* button ( ) to trigger the ILA and then switch to the ILA-hw_ila_1 tab and observe that it is waiting for the trigger to occur.

8-1-17. Press the **BTNC** (ZedBoard) or **BTN0** (Zybo) to trigger the ILA and then observe the waveform.

Notice that the LEDs are still shifting right. The ICAP output changed at 14 indicating SYNC word detected. At 19 the corrupted IDCODE came and at 32 ICAP output changed to 0xFFFFFFDB > 0xFFFFFF5B > 0xFFFFFF1B followed by CSIB signal going high indicating the reconfiguration process has been aborted.

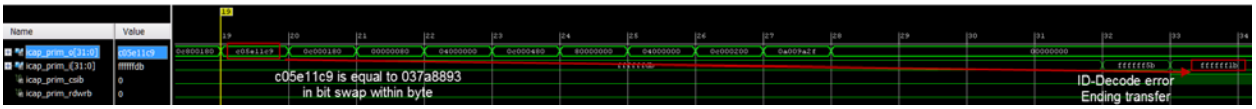


Figure 19. IDCODE word error

8-1-18. Select **File > Close Hardware Manager**

8-1-19. Power off the board and close Vivado.

Conclusion

This lab showed you how the partial reconfiguration controller (PRC) can be used with hardware events (trigger) to reconfigure RPs. The PRC has the ICAP port which can be connected to the ICAP resource. The ILA core can be used to monitor the ICAP ports and analyze the activities taking place during the reconfiguration.