Reconfiguring User Logic Using Custom ICAP Processor and Monitoring ICAP Signals Using Vivado Analyzer Lab

Introduction

In this lab, the ICAP is accessed through a provided light-weight custom IP. The custom ICAP_processor IP requires bitstream length, go, and done signals as input. The partial bitstream is provided by the processor system by reading the partial bitfiles from the SD card, storing them in the DDR memory, and sending the appropriate bitstream to the ICAP processor based on the user's selection. The design has one RP with two functional RMs. The integrated logic analyzer (ILA) core is used to monitor the ICAP signals.

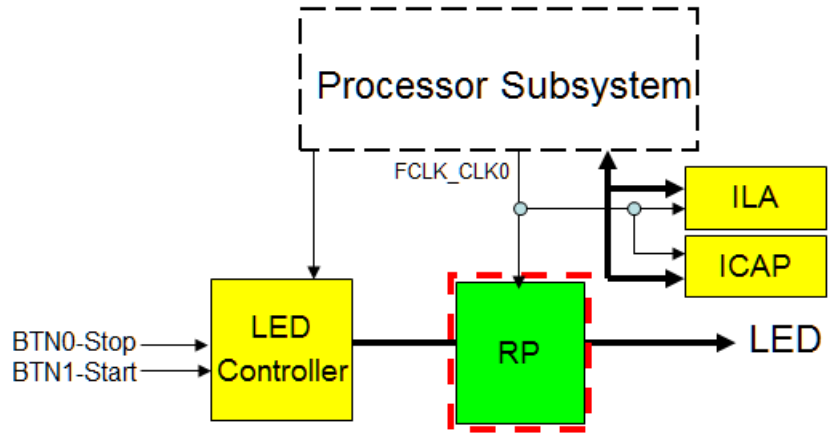
Objectives

After completing this lab, you will be able to:

- Use Tcl script to generate a Vivado IPI design having a PS7 sub-system, provided light-weight ICAP processor and ICAP processor interface IPs
- Add Integrated Logic Analyzer core to monitor ICAP ports
- Configure the ILA to perform advanced triggering and conditional triggering
- Use one Tcl script which calls 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
- Generate the corrupted partial bit files for inducing SYNC, IDECODE, and CRC errors
- Copy the generated bitstreams and the BOOT.bin on a SD Card and verify partial reconfigurable design functionality

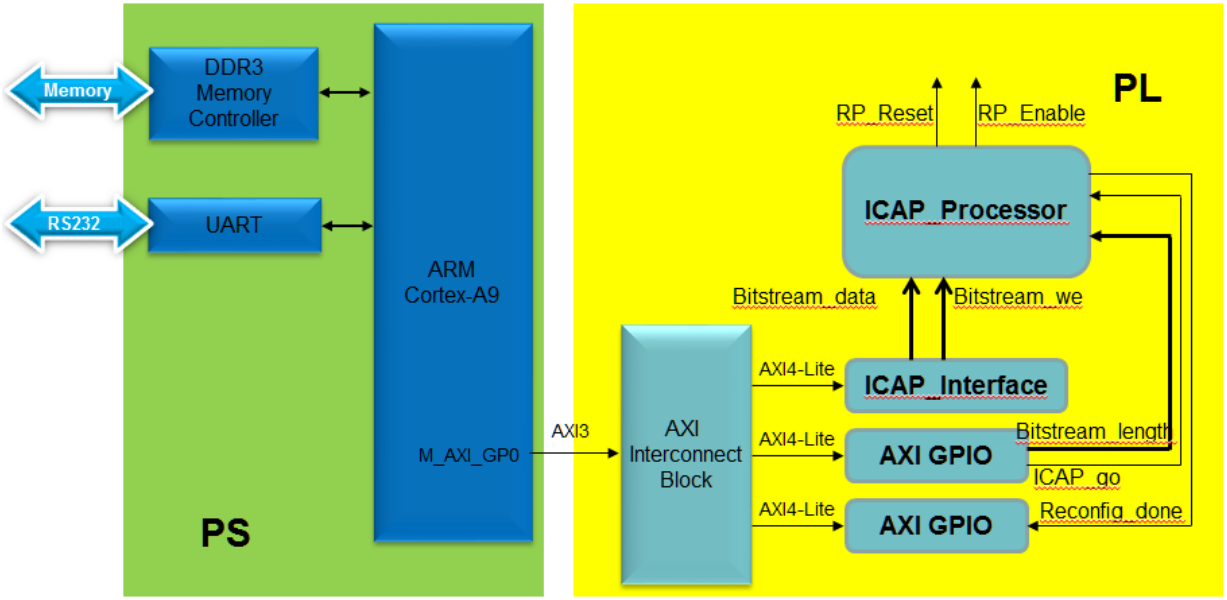
Design Description

The purpose of this lab exercise is to implement a design that is dynamically reconfigurable using the light-weight ICAP processor. The design, shown in Figure 1, consists of the processor sub-system; ICAP processor and ICAP interface IPs, ILA and one RP. The RP has two functional RMs performing right and left shifting patterns on LEDs. The dynamic partial reconfigurable modules are updated the user command

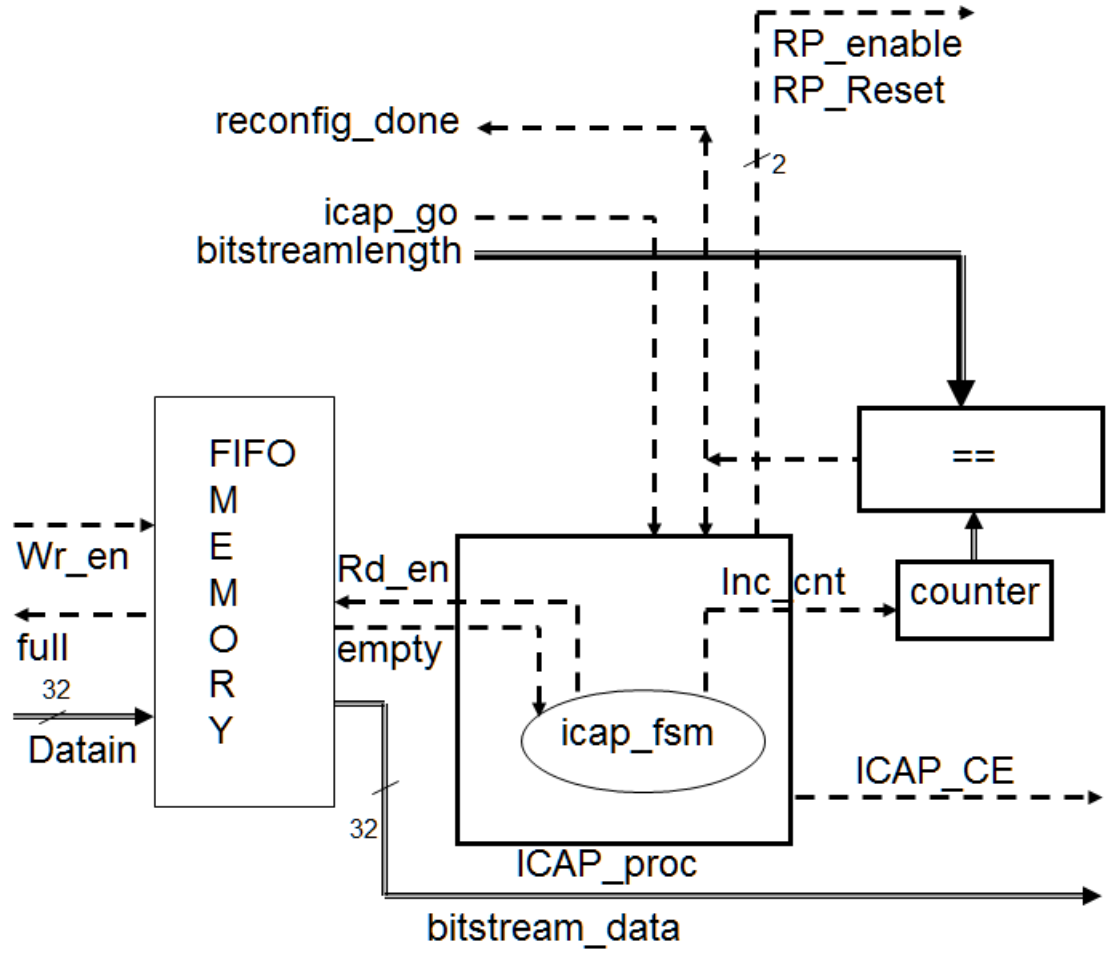


.

(a) Top-Level



(b) Processor Subsystem



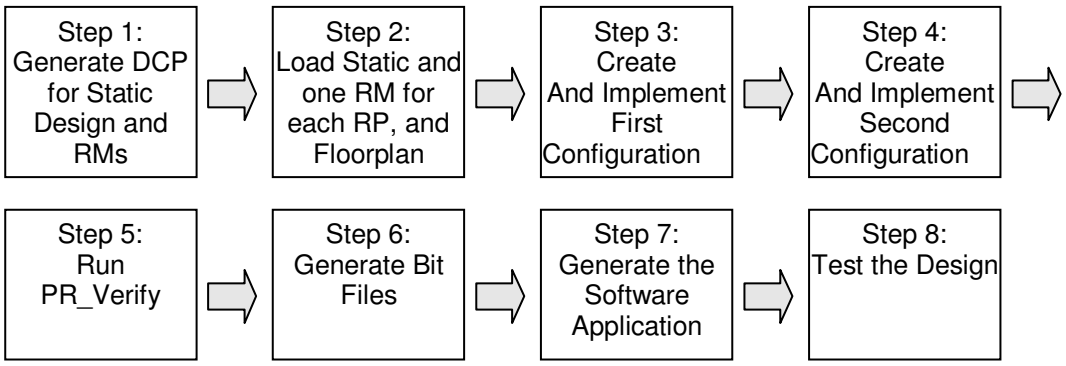
(c) ICAP_Processor IP

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 the Vivado 2014.3 program 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 2014.3 > Vivado 2014.3**

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/icap_processor_lab
```

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

```
source ps7_create.tcl
```

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

- Instantiate ZYNQ PS with *SD 0* and *UART 1* peripherals, *M_GP0* interface, and *FCLK_CLK0* and *FCLK_RESET0_N* ports enabled
- Add an instance of each of the provided *icap_processor* and *icap_interface* IPs, and two instances of AXI GPIO. Configure one GPIO instance to be 1-bit input only (axi_gpio_1 instance in the diagram) and another with both channels enabled and configured as output only: channel 1 one-bit and channel 2 32-bit wide.
- Make **FCLK_CLK0** and several other signals external so they can be monitored at one-level above by instantiating ILA
- The design looks like as shown in Figure 2.

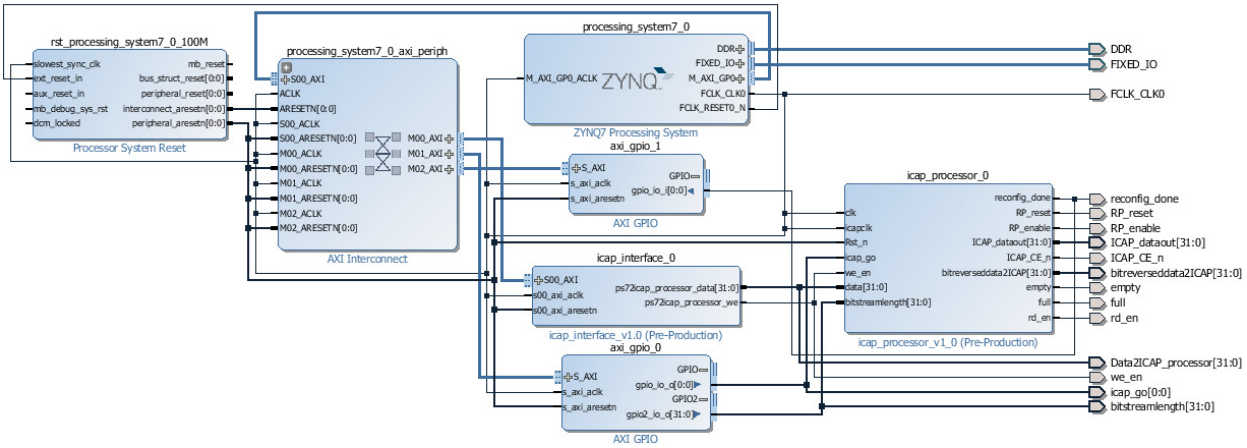


Figure 2. The processor system

- 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 the static design's rest of the modules. The design hierarchy will look like as shown in Figure 3.

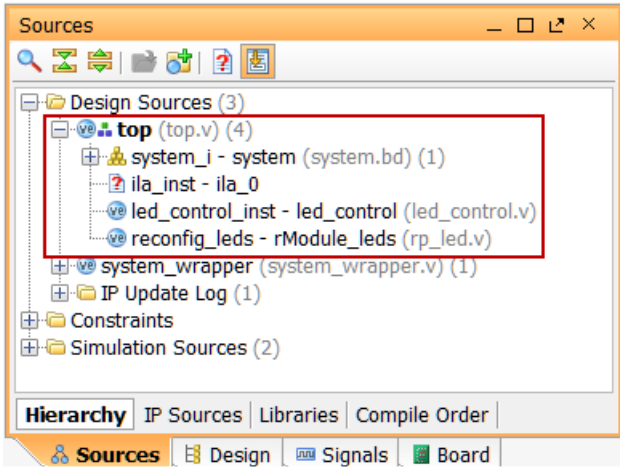


Figure 3. The design hierarchy including the processor system and other top-level modules

Notice the ? mark for the ila_inst instance as we still need to add that IP.

1-2. Add an ILA instance with the necessary number and appropriate size probes so all signals of interest can be monitored.

1-2-1. 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.

The customization window will open.

1-2-2. Set the *Number of Probes* to **13**, and *Sample Data Depth* to **2048**. Click on the check boxes of *Capture Control* and *Advanced Trigger* options as we want to utilize the functionality.

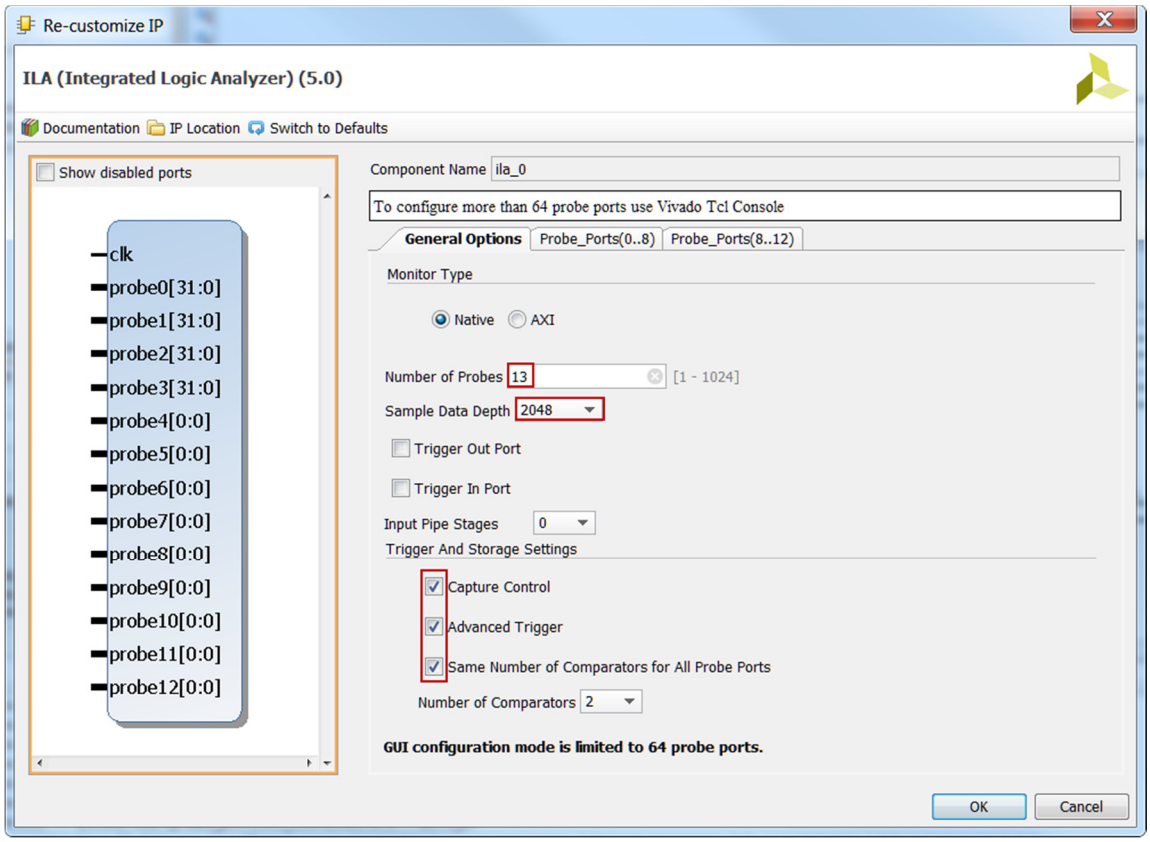


Figure 4. ILA customization- setting number of probes and other options

1-2-3. Click on the *Probes_Ports(0..7)* tab and change the size of the first four probes to **32, 32, 32, 32**, and click **OK**, leaving rest of the 9 probes width to 1.

General Options | **Probe_Ports(0..7)** | Probe_Ports(8..15)

Probe Port	Probe Width [1..4096]	Number of Comparators
PROBE0	32	2
PROBE1	32	2
PROBE2	32	2
PROBE3	32	2
PROBE4	1	2
PROBE5	1	2
PROBE6	1	2
PROBE7	1	2

Figure 5. Setting the probes widths

1-2-4. The *Generate Output Products* form will appear. Click on the **Skip** button as we will generate the output product when we synthesize the complete design.

Expand the hierarchy window and notice that there are no ? present as all required modules are in the design hierarchy.

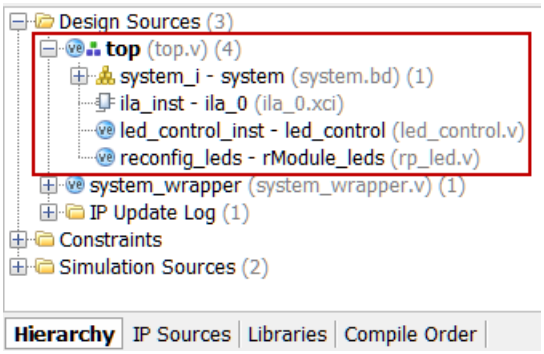


Figure 6. The design hierarchy

1-2-5. Right-click on the *ila_inst* instance in the design hierarchy and select **Out-of-Context Settings…**

1-2-6. Uncheck the *ila_0.xci* check box, click **OK** and then click **OK** again.

1-2-7. Click on the **Run Synthesis** to start the synthesis process.

When the synthesis is completed a dialog box will appear to open the synthesized design.

1-2-8. Click **Cancel**.

1-2-9. Close the project.

1-2-10. Using the Windows Explorer copy the **top.dcp** from the *c:\XUP\PR\labs\icap_processor_lab\icap_processor_lab\icap_processor_lab.runs\Synth_1* directory and place it in the *c:\XUP\PR\labs\icap_processor_lab\Synth\Static* directory.

1-3. Generate the dcp for each of the RMs.

1-3-1. In the Tcl Shell window enter the following command to change to the lab directory and hit **Enter**.

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

1-3-2. Synthesize each of the RMs (two) executing the provided Tcl script.

```
source synth_reconfig_modules.tcl
```

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. Open the **Vivado 2014.3 Tcl Shell** by selecting **Start > All Programs > Xilinx Design Tools > Vivado 2014.3 > Vivado 2014.3 Tcl Shell**

This version of tool crashes if you run the following tcl file from the Vivado GUI's Tcl Shell tab. Hence you must use the Tcl Shell in the non-GUI mode. This applies to this lab since it has ILA core in the static dcp design. Labs, not using the ILA cores, won't experience this problem.

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

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

You can enter `source run_complete.tcl` command which will do everything from next step onwards through generating bitstreams. If you use this command then read through but do not execute any of the commands until Step 7

2-1-3. Execute the following Tcl script to

```
source floorplan_design.tcl
```

The script will do the following:

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

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

- Load one RM for of 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]
  ```

- Read the debug nets information from the saved debug.xdc file

  ```
  read_xdc Synth/Static/debug.xdc
  ```

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

  ```
  write_checkpoint Checkpoint/top_link_left.dcp
  ```

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

  ```
  read_xdc Sources/xdc/fplan.xdc
  ```

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

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

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 `c:/xup/PR/labs/icap_processor_lab`.

```
source create_first_configuration.tcl
```

Note that at this point the debug cores will be generated so it will take a while to finish executing the script.

- The script will 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 ./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 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. The second command below generates the bit files with per frame CRC

```
open_checkpoint Implement/Config_left/top_route_design.dcp
set_property bitstream.general.perFrameCRC yes [current_design]
write_bitstream -bin -file Bitstreams/Config_addleft.bit
close_project
```

- Generate the bitstreams for the second configuration

```
open_checkpoint Implement/Config_right/top_route_design.dcp
set_property bitstream.general.perFrameCRC yes [current_design]
write_bitstream -bin -file Bitstreams/Config_multright.bit
close_project
```

- Generate the bitstreams with black boxes.

```
open_checkpoint Checkpoint/static_route_design.dcp
set_property bitstream.general.perFrameCRC yes [current_design]
write_bitstream -bin -file Bitstreams/blanking.bit
close_project
```

6-1-3. Close the Vivado Tcl prompt window.

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/icap_processor_lab/icap_processor_ lab*, select the *icap_processor_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\icap_processor_lab\Sources\TestApp\src* and click **OK.**

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

The program should compile successfully. Fix errors if any.

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 Zynq Boot Image.**

7-5-2. Click the Browse button of the Output BIF file path field, browse to *c:\xup\PR\labs\icap_processor_lab,* and then click **Save** with the *output.bif* 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\icap_processor_lab\icap_processor_lab\icap_processor_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\icap_processor_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\icap_processor_lab\icap_processor_lab\icap_processor_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\icap_processor_lab** and the filename is *BOOT.bin,* and click **Create Image.**

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

7-5-7. Close the project in Vivado.

Test the Design Step 8

8-1. Place the board in the SD boot mode. Make one copy each of left.bin, b_led.bin, and right.bin files. Rename the copied files as sync.bin, idcode.bin, and crc.bin respectively. Corrupt the sync.bin to have corrupted sync word, idcode.bin to have the corrupted idcode, and crc.bin to have the corrupted first frame crc. Copy them along with the BOOT.bin file on the SD card.

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/icap_processor_lab/* directory on to a SD Card.

8-1-3. Rename the partial bitstreams in the `bitstreams` directory as listed in the table.

Source Name	New Name
blanking_pblock_reconfig_leds_partial.bin	b_led.bin
Config_left_pblock_reconfig_leds_partial.bin	left.bin
Config_right_pblock_reconfig_leds_partial.bin	right.bin

8-1-4. Make one copy each of *left.bin*, *b_led.bin*, and *right.bin* files. Rename the copied files as **sync.bin, idcode.bin,** and **crc.bin** respectively.

8-1-5. Using a Hex Editor, open the **sync.bin** file and make change to the SYNC word so it looks like as shown below and then save the file.

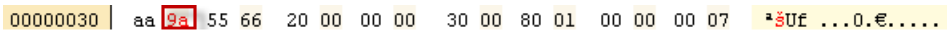


Figure 7. Corrupting the sync word

8-1-6. Similarly, open the **idcode.bin** file, change the IDCODE field as shown below, and save it.

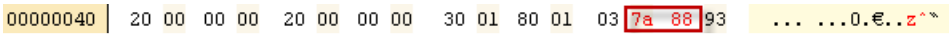


Figure 8. Corrupting the id code

8-1-7. Similarly, open the **crc.bin** file, change the crc code of the first frame as shown below (note the file offset [row starting at 210]), and save it.

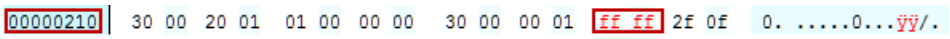


Figure 9. Corrupting the first frame's crc

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

8-1-9. Copy the partial bitfiles (the original b_led, Left, right, and the new sync, idcode, and crc files) to the SD card.

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

8-2. Connect the board with one micro-USB cable to the PROG UART connector. 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.

8-2-1. Make sure that one micro-usb cable is connected between the **PROG UART** connector of the board and the PC.

8-2-2. Power ON the board.

8-2-3. Start a terminal emulator program such as TeraTerm or HyperTerminal.

8-2-4. Select the appropriate COM port (you can find the correct COM number using the Control Panel).

8-2-5. Set the *COM* port for **115200** baud rate communication.

8-2-6. Press **BTN7** to display a menu.

8-2-7. Follow the menu and test various reconfigurations.

8-3. Analyze waveforms using Vivado Analyzer.

8-3-1. Make sure that the working directory in the Tcl shell is `c:/xup/PR/labs/icap_processor_lab`. If not then set it using the **cd** command.

8-3-2. Enter the following command to open the hardware manager, program the FPGA, and open the hardware manager's dashboard.

```
source run_ila.tcl
```

8-3-3. Click the *Stop Trigger* button ( ) to see the waveform.

8-3-4. Press **BTN7** on the board, wait for the Done LED to illuminate, and then 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.

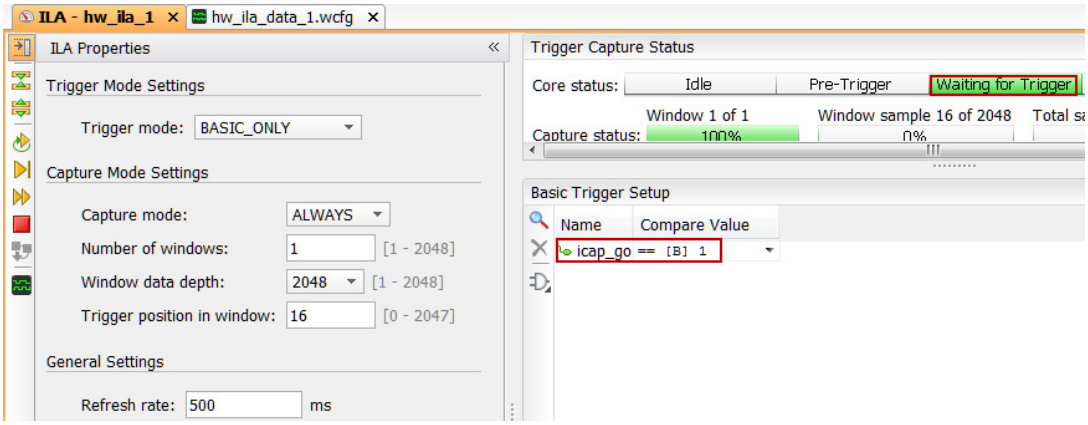


Figure 10. Waiting for the trigger condition to occur

8-3-5. In the terminal window, type **L** and observe the LEDs are shifting left and the ILA has triggered.

8-3-6. Zoom in into the beginning part of the waveform, click around 380 and notice the SYNC word is detected and the ICAP output changes from 0xFFFFFF9B to 0xFFFFFFDB around sample *373* indicating SYNC word is detected.

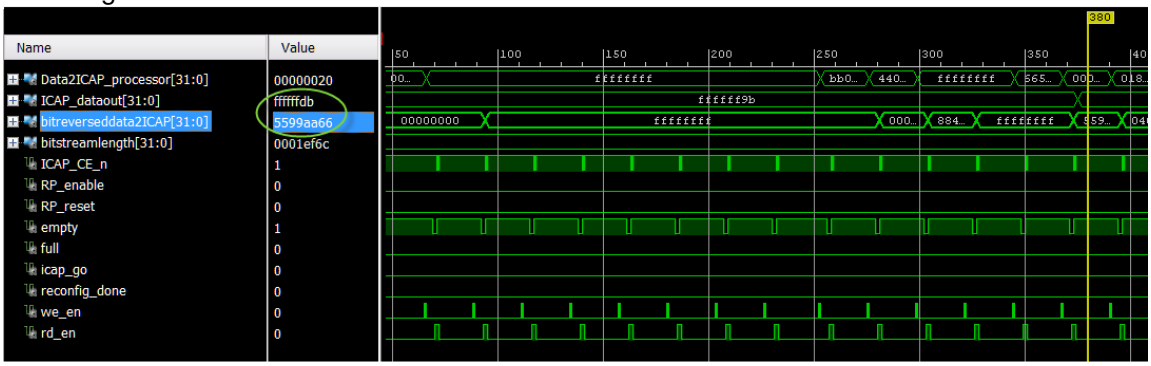


Figure 11. SYNC word detected

8-3-7. View the first few samples to see that the *RP_enable* is de-asserted when *icap_go* is asserted. You want to isolate the RM when the reconfiguration is going on. You can use this signal (*RP_enable*) in your RM interface logic to isolate the RP during the reconfiguration.

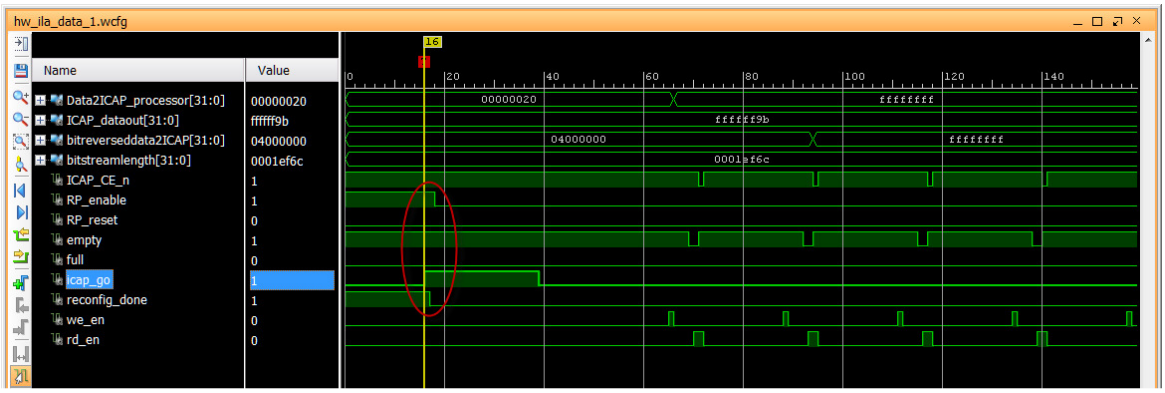


Figure 12. Activities around starting of the reconfiguration process

8-3-8. 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.

8-3-9. Type **S** to send the SYNC word corrupted bitstream and observe the waveform.

Notice that the LEDs are still shifting left, however at 380 the ICAP output did not change.

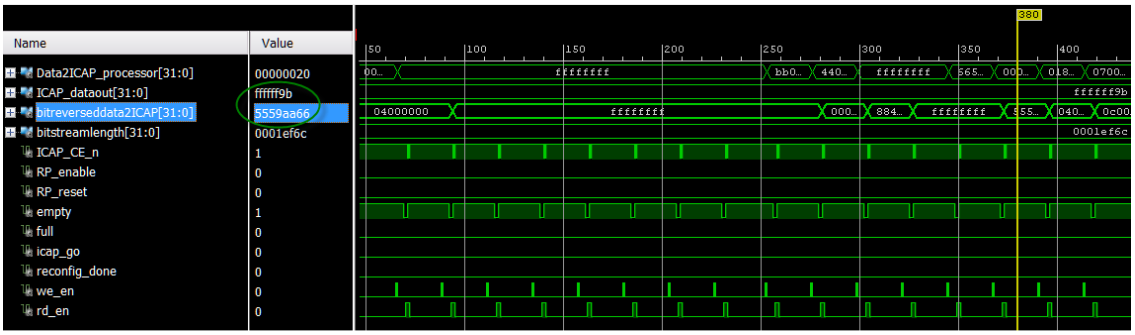


Figure 13. SYNC word error

8-3-10. Type **R** and observe the LEDs shifting right.

8-3-11. 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.

8-3-12. Type **I** to trigger the ILA and then observe the waveform.

Notice that the LEDs are still shifting right. The ICAP output changed around 373 indicating SYNC word detected. Around 523 the corrupted IDCODE came and around 800 the ICAP output changed to 0xFFFFFF5B followed by status change to 0xFFFFF1B indicating the reconfiguration was aborted.

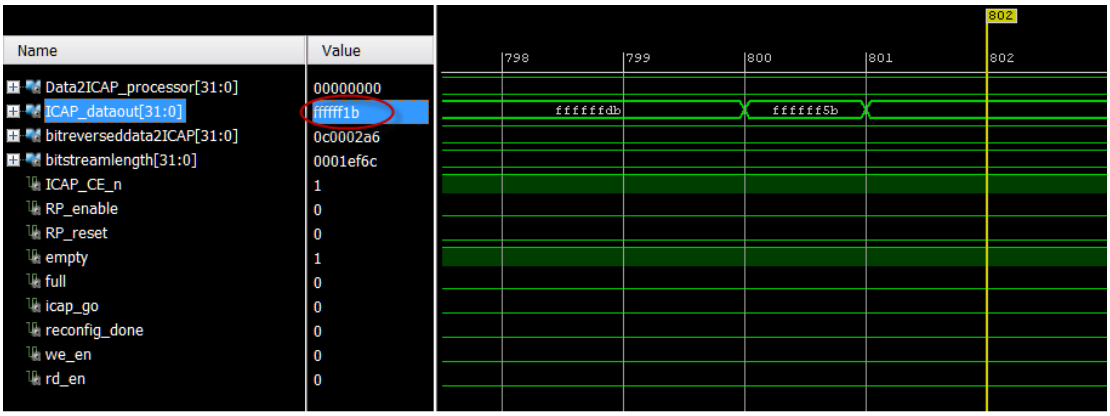


Figure 14. IDCODE word error

8-4. Use Advance Triggering to analyze end of the configuration activities

8-4-1. In the *ILA – hw_ila_1* tab, click on the drop-down button of the *Trigger mode* and select **ADVANCED_ONLY**.

8-4-2. Click on the browse button, browse to `c:/xup/PR/labs/icap_processor_lab/Sources` and select the provided **ila.tsm** (the trigger state machine).

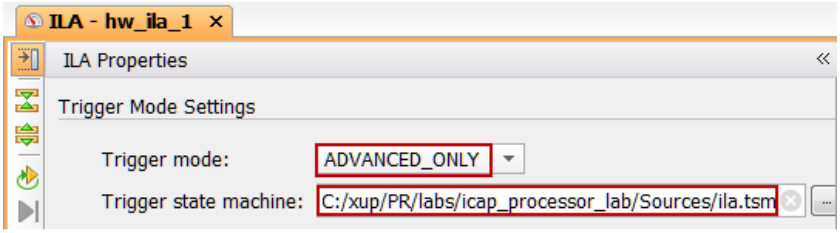


Figure 15. Setting the ILA for the advanced triggering

The ila.tsm will be loaded and the window will open showing the state machine.

C:/xup/PR/labs/icap_processor_lab/Sources/ila.tsm

```
1 state  wait_for_icapgo:
2   if ((icap_go == 1'b1)) then
3     goto  wait_for_reconfig_done;
4   else
5     goto  wait_for_icapgo;
6   endif
7
8   state  wait_for_reconfig_done:
9     if ((reconfig_done == 1'b1)) then
10      trigger;
11    else
12      goto  wait_for_reconfig_done;
13    endif
14
```

Figure 16. The ILA state machine

Line 1 and 8 define states. In the *wait_for_icapgo* state, the ILA will wait for the *icap_go* to become **1** and when the condition occurs, it will go to the second state- *wait_for_reconfig_done*. Once in the *wait_for_reconfig_done* state, it will wait for the *reconfig_done* to become **1**. When the condition occurs it will trigger storing the number of pre-trigger samples and filling the rest of the buffer with the post-trigger samples.

8-4-3. Set the Trigger position to 2000 (as we are interested in what was happening before the reconfiguration is completed).

8-4-4. Click on the *Run Trigger* button to arm the ILA and waiting for user input in the terminal window.

8-4-5. In the terminal window type **l**, **r**, or **b** to successfully reconfigure the RM.

The ILA will trigger. Switch to the waveform and view the end area. Notice that the ICAP_dataout changes from 0xFFFFFFDB (normal reconfiguration) to successfully completed reconfiguration (0xFFFFFF9B).

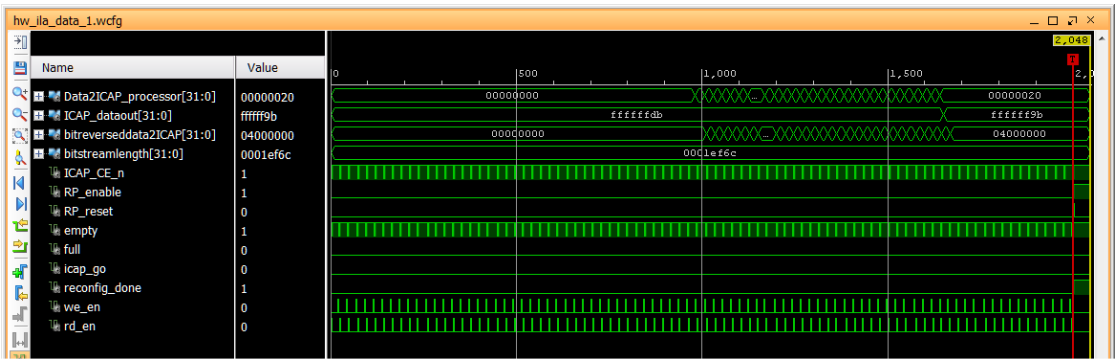


Figure 17. The triggered ILA waveform view

8-4-6. Zoom to the end of the capture (1900 – 2048 samples) and observe the activities.

Notice that after the *reconfiguration_done* goes high, the *RP_enable* is asserted along with one-clock cycle *RP_reset* pulse. This enables the brought-in RM and also resets it to the starting desired state.

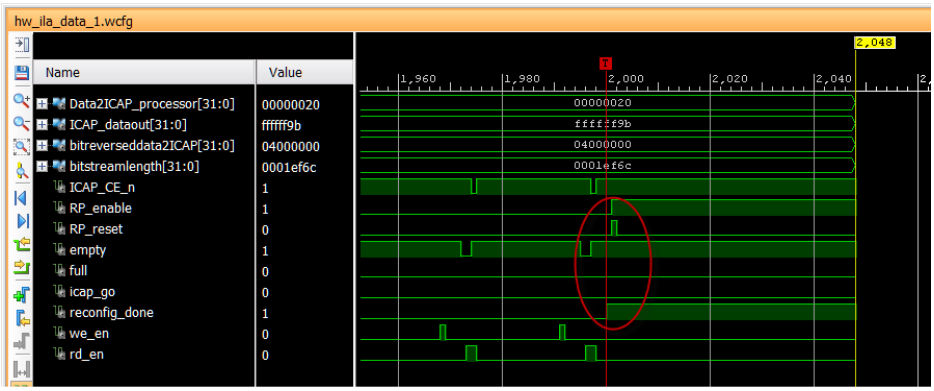


Figure 18. Zoomed view showing the activities on various signals when the reconfiguration is done

8-5. Use Advance Triggering to analyze the crc error

8-5-1. In the *ILA – hw_ila_1* tab, click on the browse button of the *Trigger state machine*, browse to `c:/xup/PR/labs/icap_processor_lab/Sources` and select the provided **ila_crc.tsm**.

Observe that the second state is monitoring ICAP_dataout and waiting for 32'HFFFFF1B (the abort word).

8-5-2. Set the trigger position to **1024**.

8-5-3. Run the ILA.

8-5-4. Press **L** to configure RM with the left shift functionality. Notice that the ILA did not trigger since the abort sequence did not occur.

8-5-5. Press **C** to configure RM with the corrupted CRC, and observe the ILA has triggered.

The ICAP_dataout changes value from 0xFFFFFFDB > 0xFFFFFF5B > 0xFFFFFF1B, i.e. sync received to configuration error.

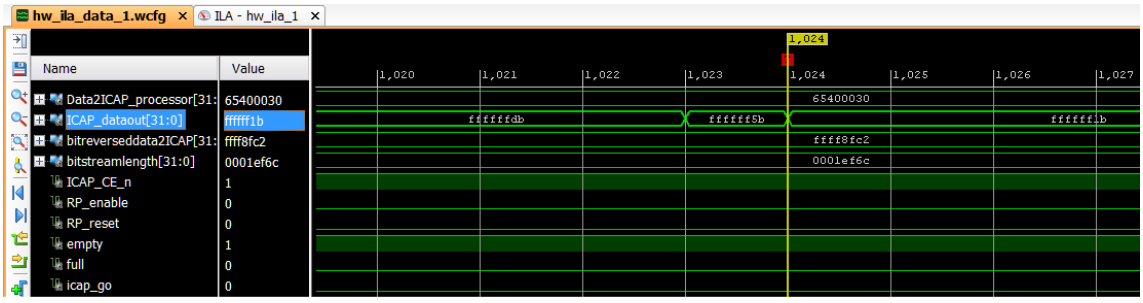


Figure 19. CRC error

8-5-6. If you scroll left (around sample 1009), you will see 0x43F1FFFF, the corrupted CRC, on the DATA2ICAP_processor bus.

8-5-7. In the **Hardware** window, select the *Zynq* device and look at its properties.

8-5-8. In the *Properties* form, expand the **CONFIG_STATUS** register and note that the *BIT00_CRC_ERROR* has value of **1**.

If you don't see it to be 1 then right-click on the Zynq device and select Refresh Device. The status register will be updated.

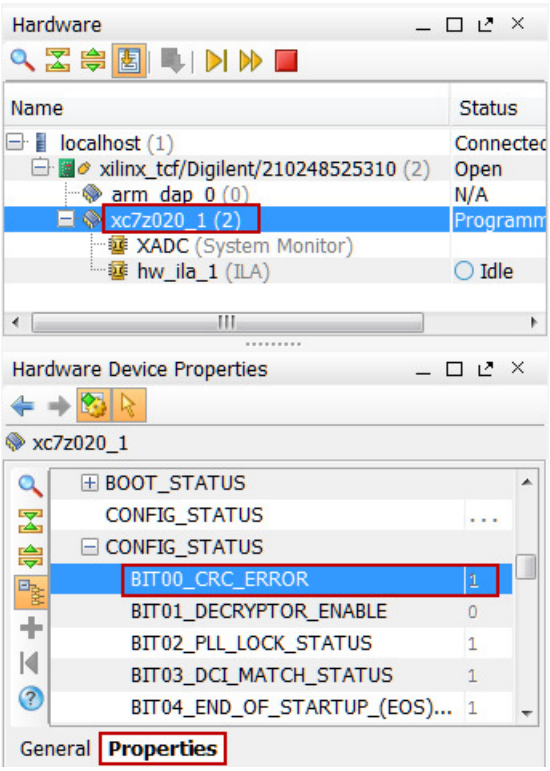


Figure 20. Verifying the CRC error in the CONFIG_STATUS register

8-5-9. Select **File > Close Hardware Manager**

8-5-10. Power off the board and close Vivado.

Conclusion

This lab showed you how the custom ICAP_processor can be used to reconfigure RPs. The ILA core was used to monitor the ICAP ports and analyze the activities taking place during the reconfiguration including various error conditions. You also used advanced triggering features of the ILA.