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

Introduction

In this lab, you will go through the project-based PR flow methodology. The design consists of the ICAP 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.

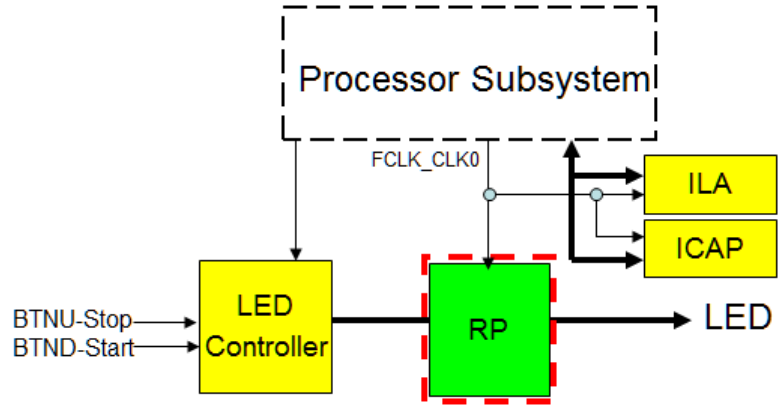
Objectives

After completing this lab, you will be able to:

- Use project-based PR flow methodology
- 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
- Enable the PR-flow methodology
- Create partition
- Use PR flow wizard to create multiple configurations
- Synthesize, implement, and generate bitstream from the project
- 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

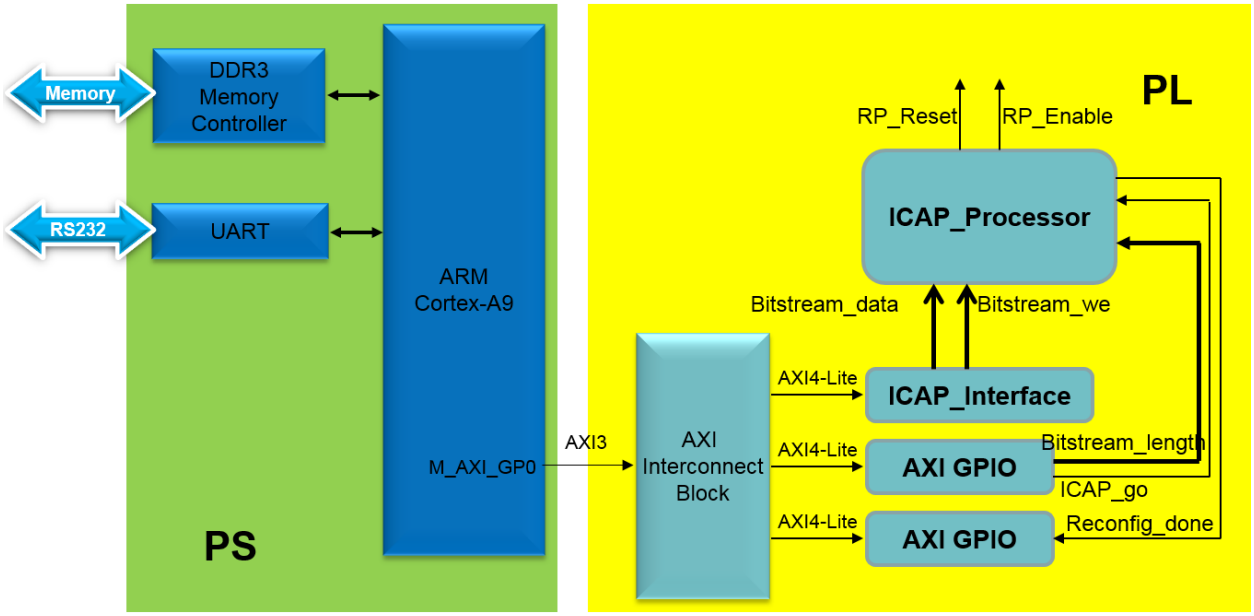
Design Description

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

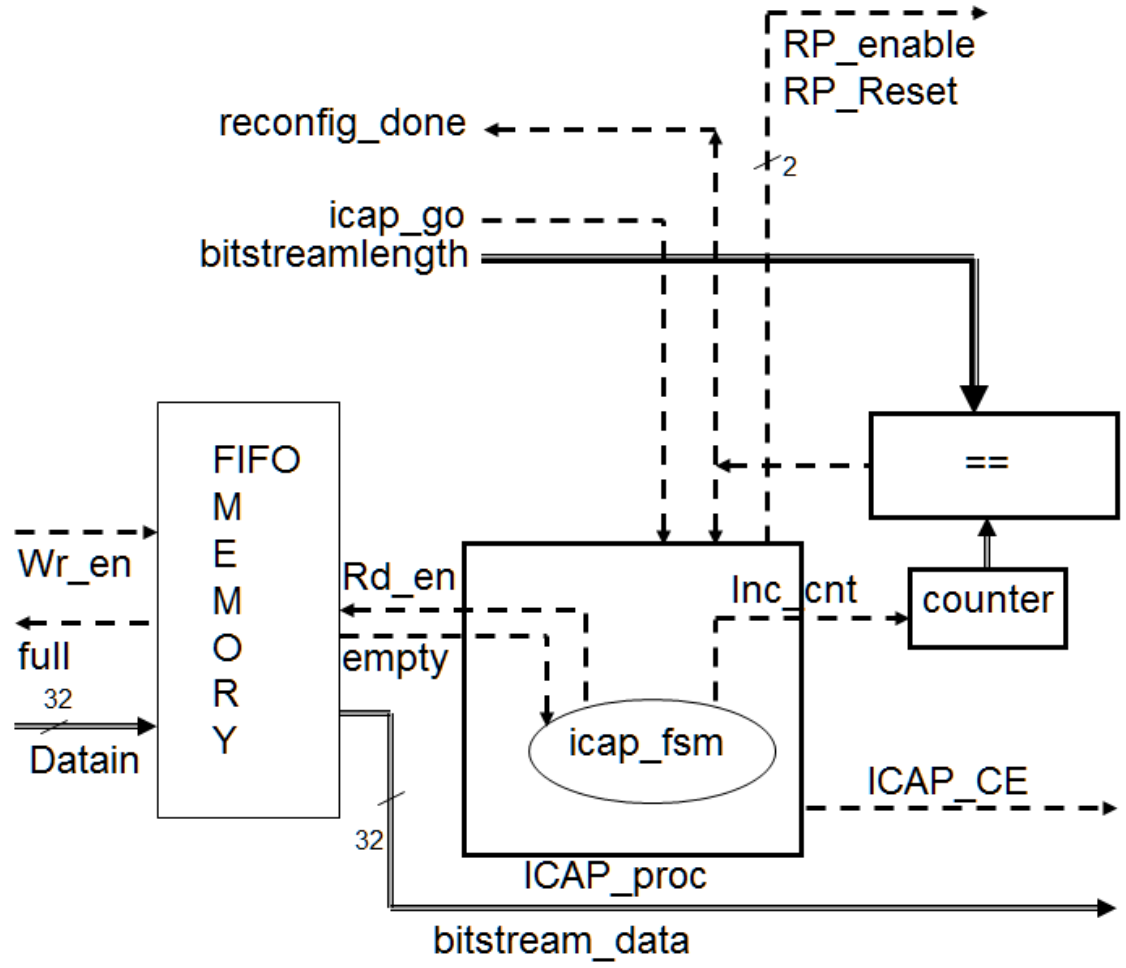


.

(a) Top-Level



(b) Processor Subsystem



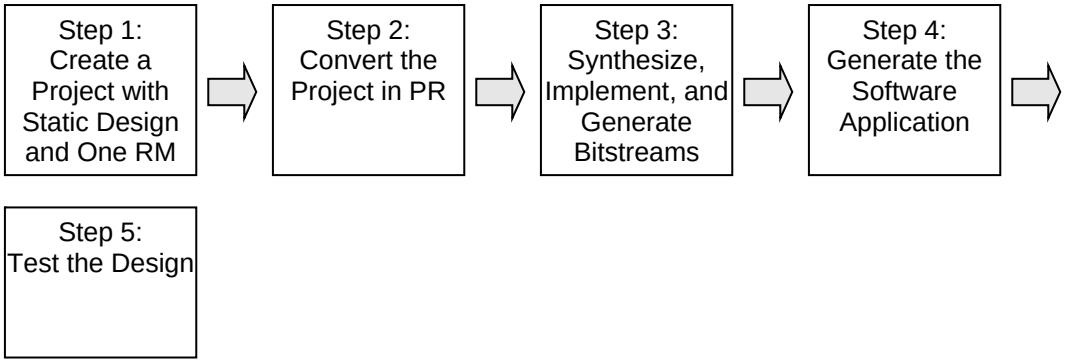
(c) ICAP_Processor IP

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



Create A Project with Static Design and One RM — Step 1

1-1. Start the Vivado 2016.3 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 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/icap_processor_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 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.

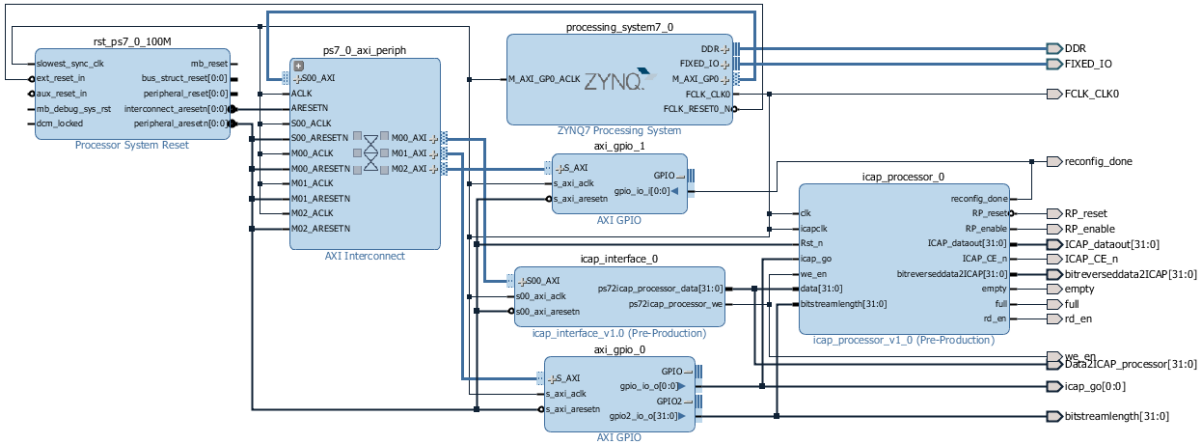


Figure 2. The processor system

- 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.

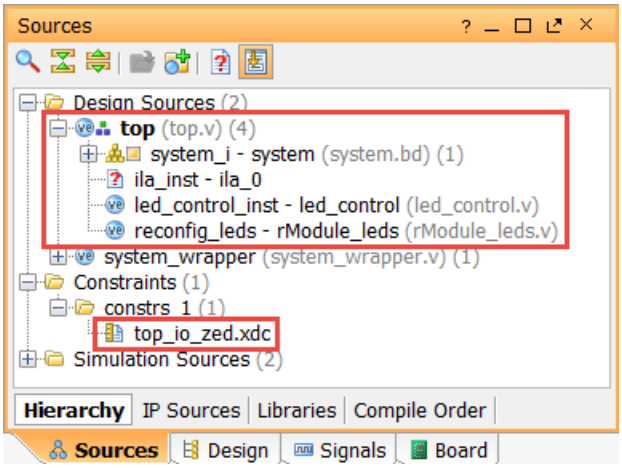


Figure 3. The design hierarchy including the processor system and other top-level modules

Notice the "?" mark for the ila_inst instance. We will add the ila_inst next. Also notice that the top-level IO constraints are provided through the xdc file.

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. Click **IP Catalog** under *Flow Navigator > Project Manager*.

1-2-2. 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.

1-2-3. 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.

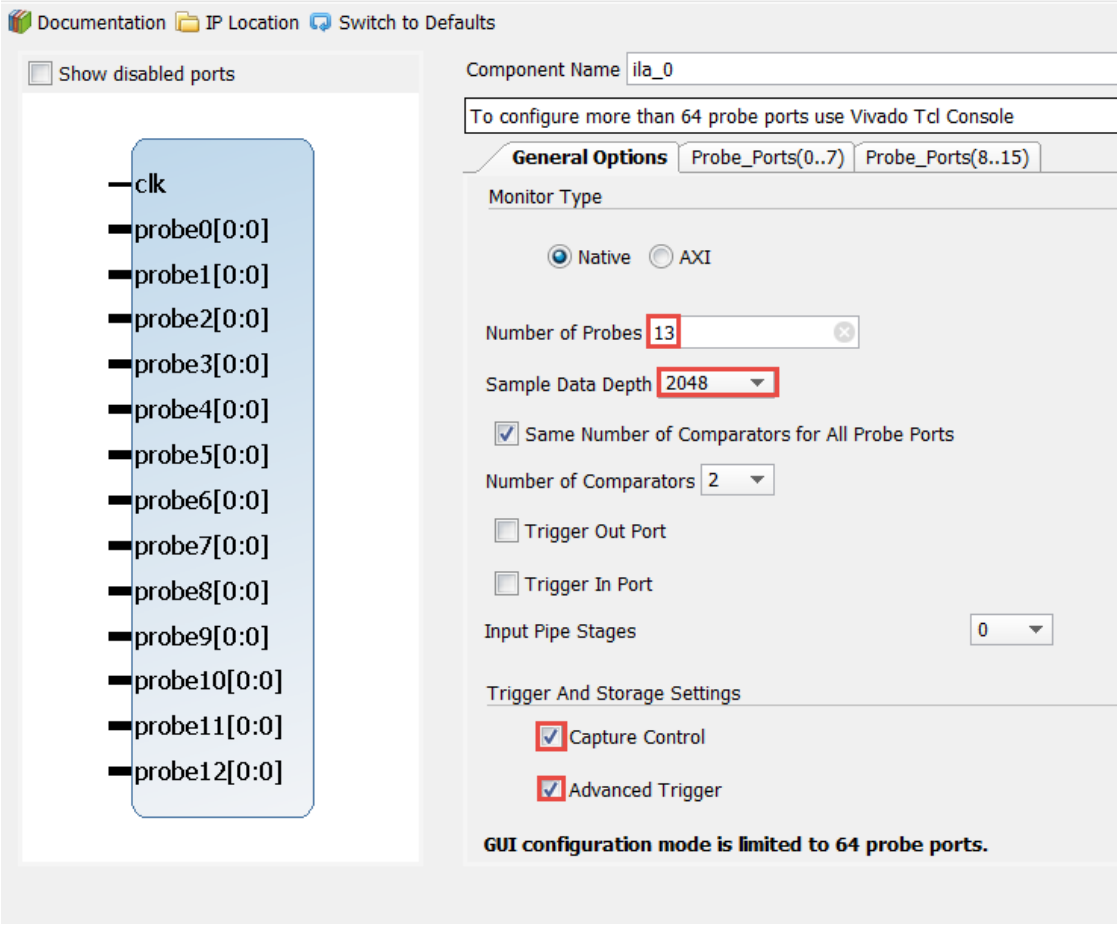


Figure 4. ILA customization- setting number of probes and other options

1-2-4. 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.

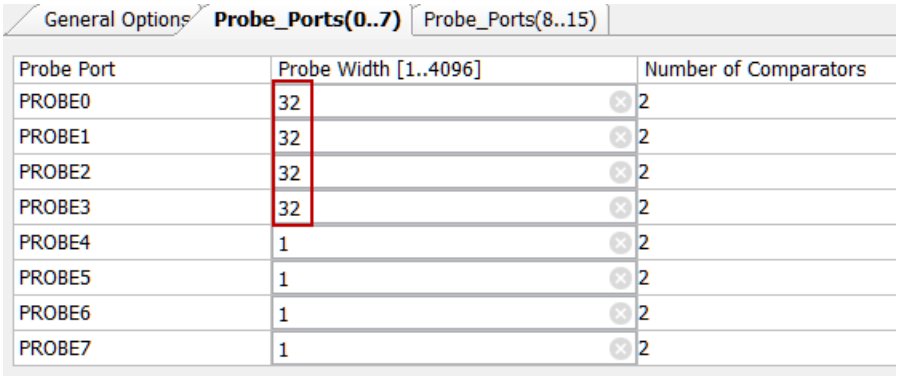


Figure 5. Setting the probes widths

1-2-5. The *Generate Output Products* form will appear. Select *Global* and then click **Apply**.

1-2-6. 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 we have all the modules defined.

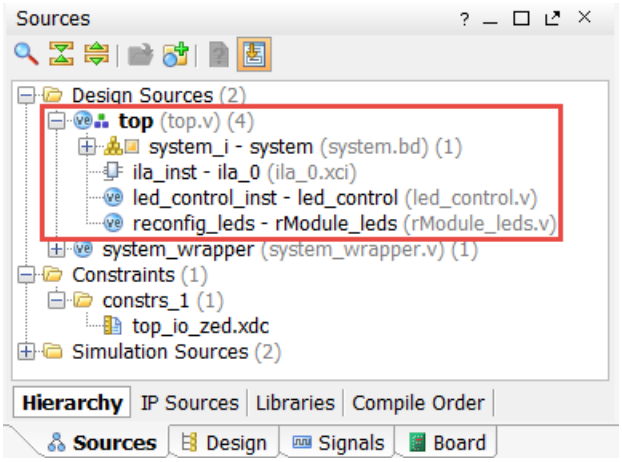


Figure 6. The design hierarchy

At this stage, the standard project has been created. Next we will enable the project-based PR flow.

Convert the Project into PR — Step 2

2-1. Enable project-based PR flow. Define partition and RM.

2-1-1. Select **Tools > Partial Reconfiguration Wizard…** to enable the project-based PR flow.

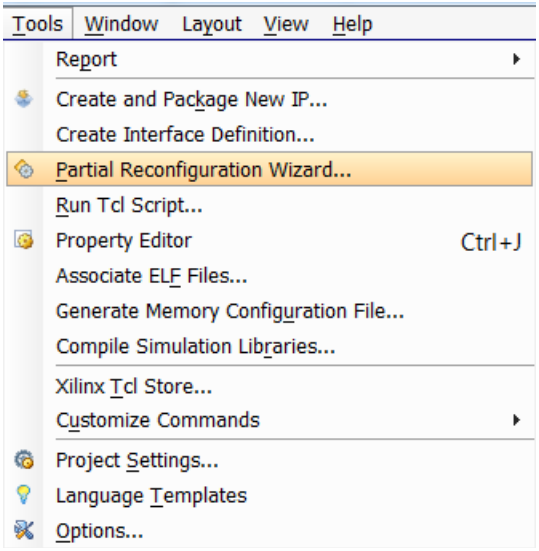


Figure 7. Enabling the project-based PR flow

Once this is set it cannot be undone, so archive your project before selecting this option.

2-1-2. In the ensuing dialog box, click **Convert** to turn this project into a PR project.

Note that the **Partial Reconfiguration Wizard** button is now available under the *Project Navigator* in the **Flow Navigator** tab.

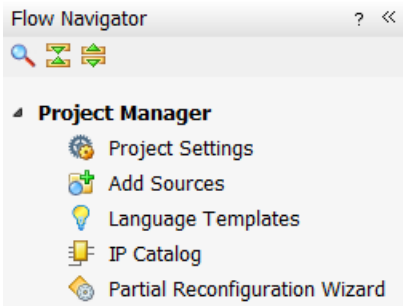


Figure 8. Project Reconfiguration Wizard process available in Project Manager

2-1-3. Right-click on the *reconfig_leds* instance and select the **Create Partition Definition…** option.

2-1-4. In the dialog box that appears, enter **reconfig_leds** in the Partition Definition Name field, change the Reconfigurable Module Name to **left** since that functionality is defined in the initial design. Click **OK**.

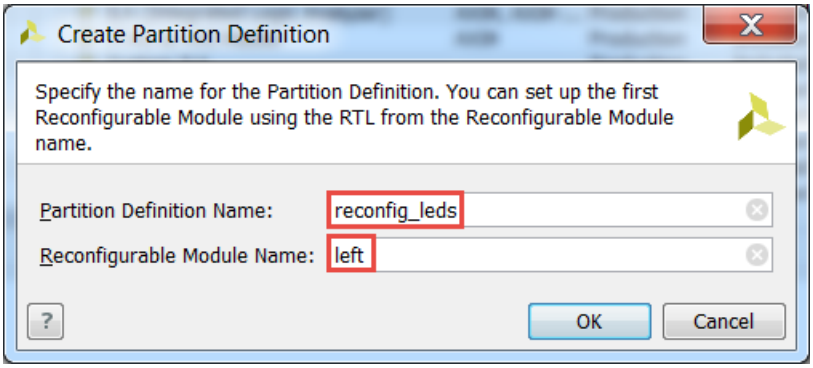


Figure 9. Defining partition name

Notice the Diamond icon in front of the reconfig_leds indicating that it is a reconfigurable partition. Also notice that the Partition Definitions tab is now available, showing the list and contents of all Partition Definitions (just one at this point) in the design.

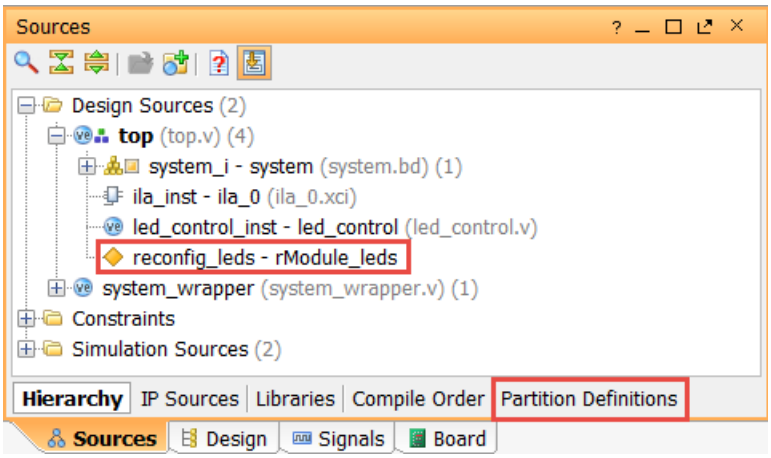


Figure 10. Reconfigurable partition and the Partition Definitions tab

2-1-5. Select the *Partition Definitions* tab and observe that **left** RM is defined having source files.

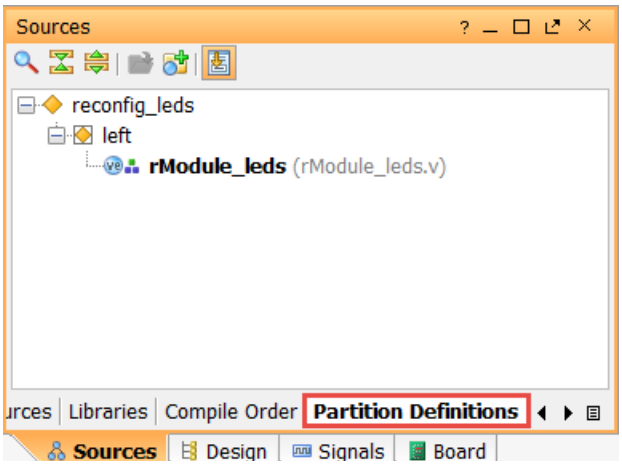


Figure 11. The Partition tab

2-2. Use Partial Reconfiguration Wizard to define and add more RMs.

2-2-1. Launch the **Partial Reconfiguration Wizard** by selecting this option under the Tools menu or from the *Flow Navigator*.

2-2-2. Click **Next** to get to the **Edit Reconfigurable Modules** page.

Here you can add, remove and edit RMs using the buttons on the left hand side of the page. Notice that **left** RM is already included since we have defined it.

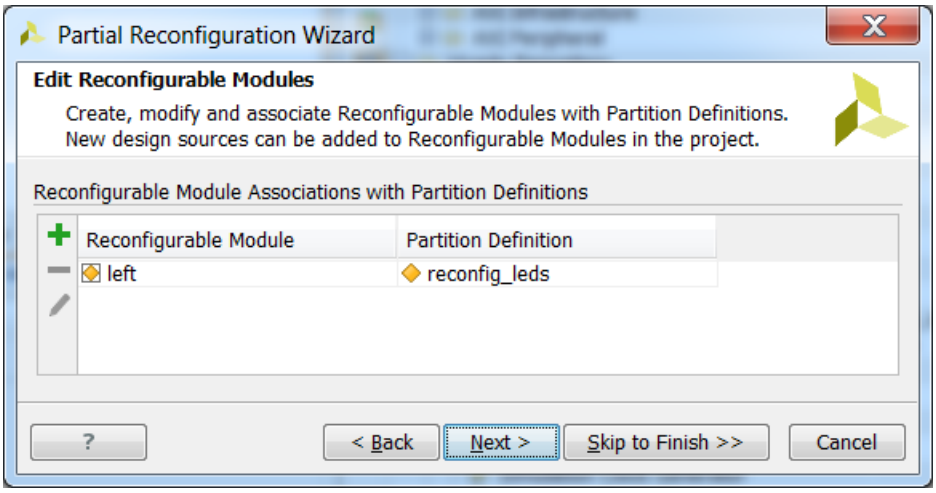


Figure 12. Edit Reconfigurable Modules dialog box

2-2-3. Click on the green + icon to add a new RM.

2-2-4. Enter **right** in the *Reconfigurable Module Name* field, browse to *c:\xup\PR\labs\icap_processor_lab\Sources\rModule_leds\rightshift* directory, select the **rModule_leds.v** file and click **OK.**

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

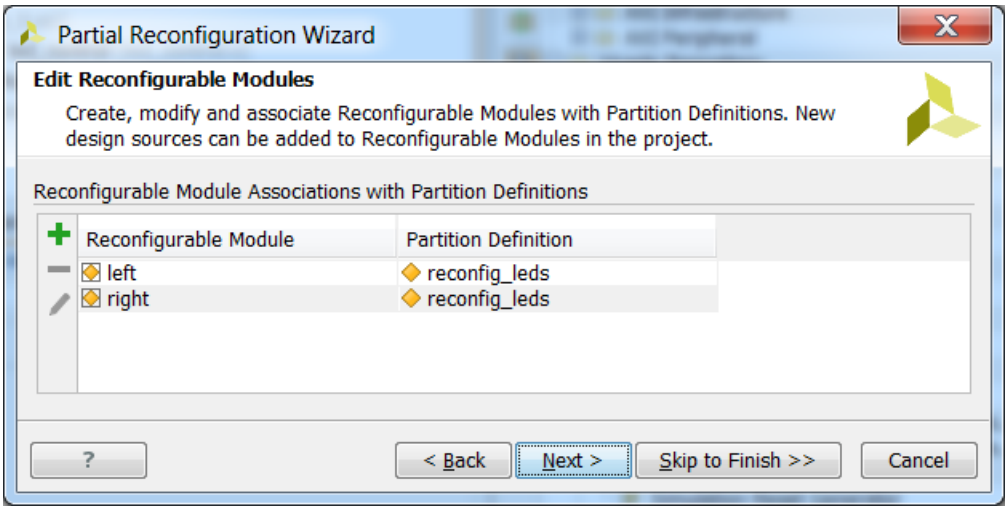


Figure 13. Created RMs

2-2-6. Click **Next** to go to the **Configurations** page since we have defined all the RM.

2-3. Continue with the wizard to create configurations.

2-3-1. Let the Wizard create the configurations by selecting the **automatically create configurations** link.

You will see two configurations listed with default names. You can make changes to these names if you desire by double-clicking on the Name field and typing over.

2-3-2. Change *config_1* to **config_left**, and *config_2* to **config_right** as meaningful names.

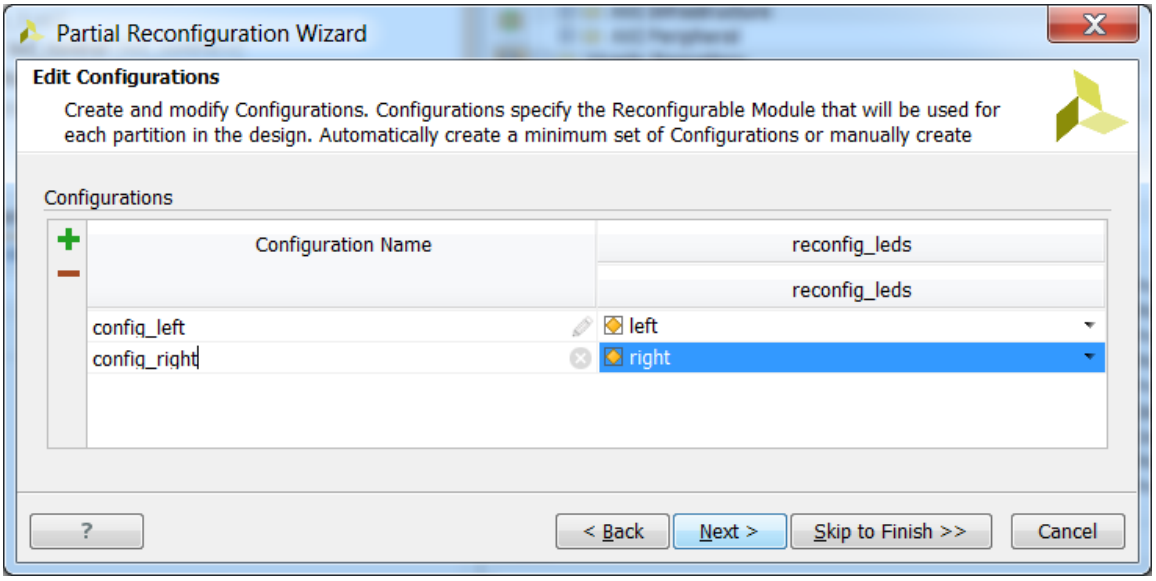


Figure 14. Configurations created by the wizard

2-3-3. Click on the green + icon to add another configuration.

2-3-4. Enter **config_blank** as the name and hit Enter.

The Edit Configuration Runs window will show-up.

2-3-5. Click **Back** button to see the *Edit Configurations* page.

2-3-6. Click on the drop-down button of the *config_blank* and select **<greybox>**.

This module defines the input and output but won't have any functionality, i.e. it will be a blackbox.

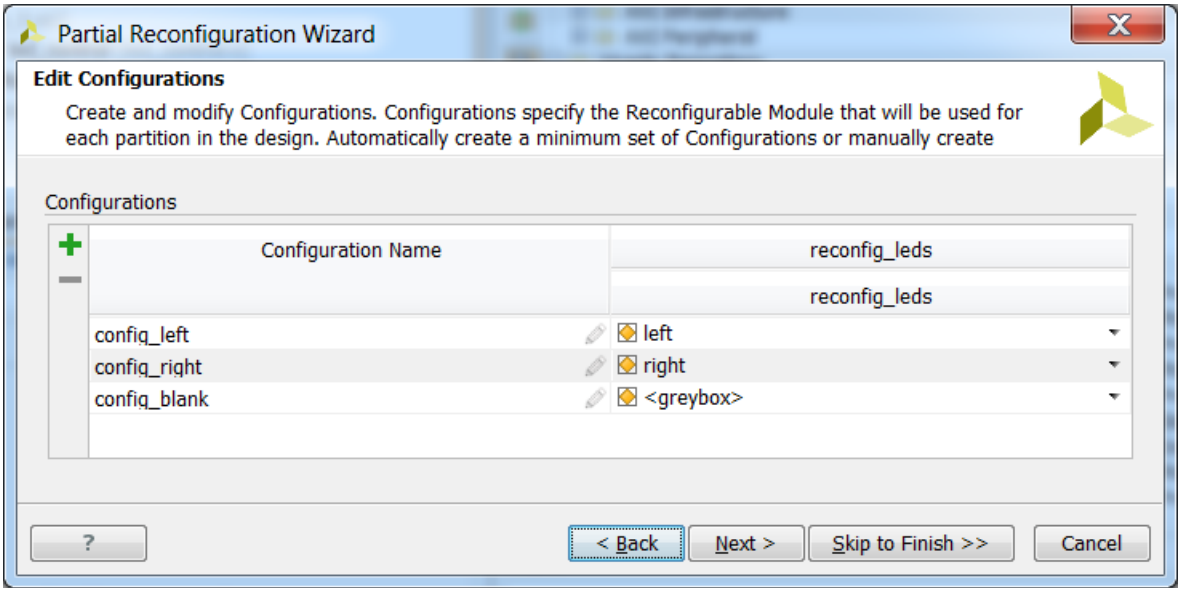


Figure 15. Blank configuration added

2-3-7. Click **Next** to get to the **Edit Configuration Runs** page.

As with configurations themselves, the runs used to implement each configuration can be automatically or manually created. A parent-child relationship will define how the runs interact – the parent run implements the static design and all RMs within that configuration, then child runs reuse the locked static design while implementing the RMs within that configuration in that established context.

2-3-8. Click on the **automatically create configuration run** link to populate the Configuration Runs page with the minimum set of runs.

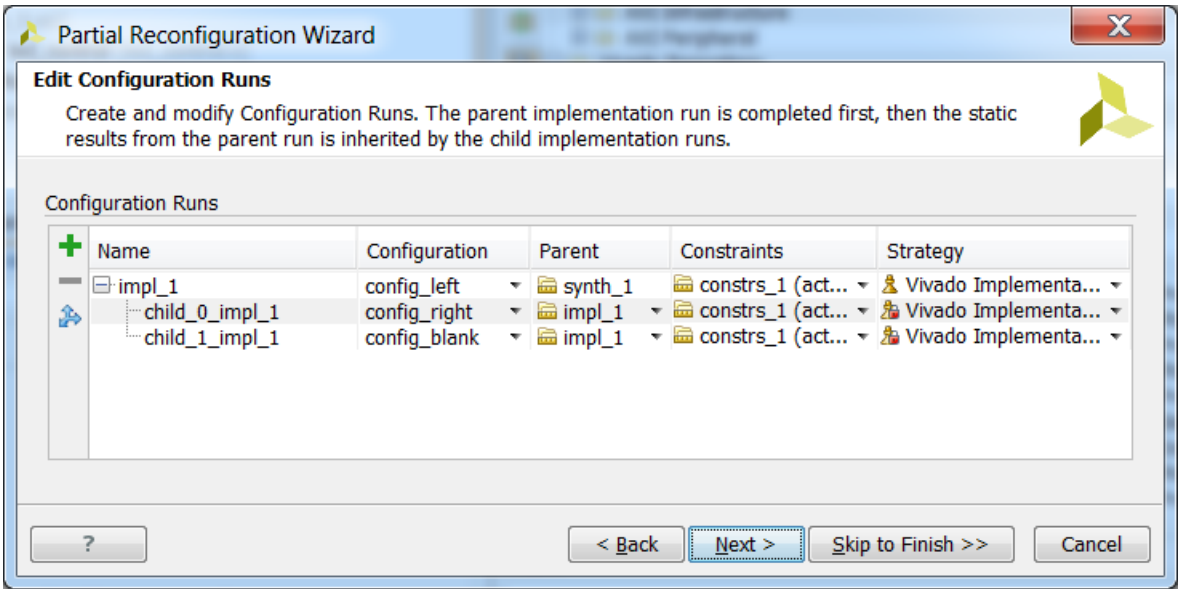


Figure 16. Configuration runs

2-3-9. Click **Next** to see the Summary page then **Finish** to complete the design setup and exit the Wizard.

Note that the left and right RMs will be synthesized in Out-of-Context mode. The static design will be synthesized first having left shift functionality. The results of which will be used to create partial bitstreams of config_right and config_blank

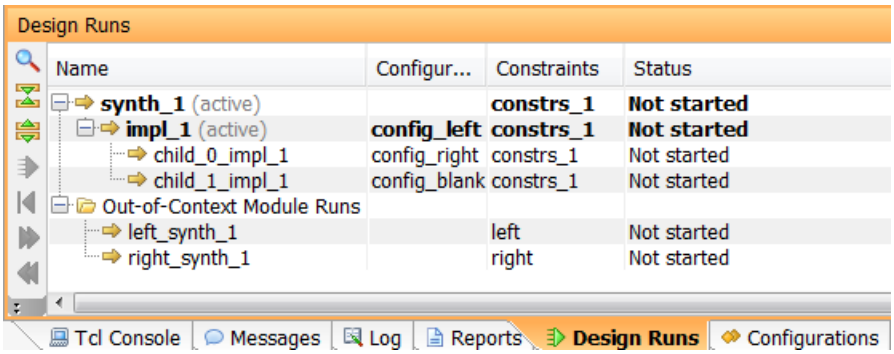


Figure 17. Created design runs

Synthesize, Implement and Generate Bitstreams Step 3

3-1. Synthesize the design.

3-1-1. 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. At this point you can either add a constraint file which defines the pblocks or you can floorplan the design.

3-2. Click Cancel if you want to use the pre-created floorplan xdc file otherwise go to 3-3.

3-2-1. Click **Cancel**.

3-2-2. Click on **Flow Navigator > Project Manager > Add Sources**

3-2-3. Select the **Add or Create Constraints** option.

3-2-4. Click **Next** and then click on the Add Files button.

3-2-5. Browse to *c:\XUP\PR\labs\icap_processor_lab\Sources\xdc,* select **fplan_zed.xdc** (for ZedBoard) or **fplan_zybo.xdc** (for Zybo) and click OK. Make sure that Copy constraints files into the project is checked.

3-2-6. Click **Finish** to add the constraint file.

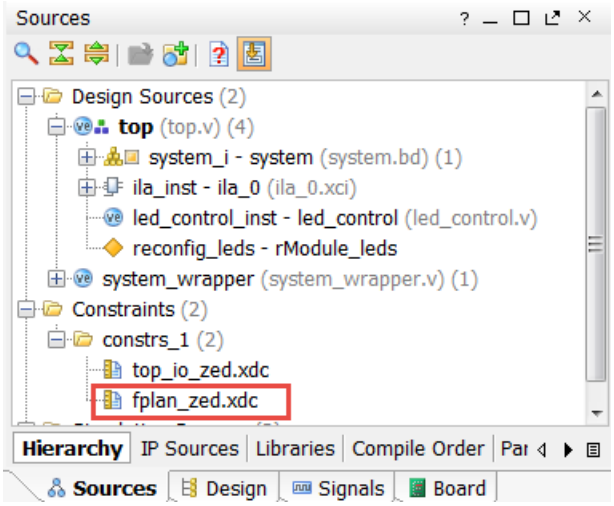


Figure 18. Floorplan constraints added to the project

3-2-7. Skip to **3-5**.

3-3. Skip this and go to 3-5 if you have already done 3-2 above, otherwise click on the Open Synthesized Design and click OK.

3-3-1. Select the *Open Synthesized Design* option and click **OK** to open the design.

3-3-2. Select **Layout > Floorplanning**.

3-3-3. Select **Edit > Find**. In the *Find* field. Select **Sites** in the *Find* drop-down box.

3-3-4. Ensure *Name* and *contains* are selected, and in the text box change * to ***SLICE_X16Y20**.

3-3-5. Click on the **+** button, then select **OR** using the drop-down button, choose *Name contains* again, type ***SLICE_X19Y24**, and click **OK**.

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

3-3-6. 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.

3-3-7. Select the **reconfig_leds** instance in the *Netlist* window, right-click, and select *Floorplanning* > *Draw Pblock.*

3-3-8. Draw a box that bounds `SLICE_X16Y20:SLICE_X19Y24` marked in the previous step.

3-3-9. Click **OK** to include SLICE (20 slices) as the resources to be reconfigured.

3-3-10. Select the **reconfig_leds** instance, select the *properties* tab, check the *RESET_AFTER_RECONFIG* property, and select **ON** for the *SNAPPING_ON* property.

3-3-11. Click File > Save Constraints and click OK.

3-3-12. Select *Create a New File* option and enter **fplan_zed.xdc** (for ZedBoard) or **fplan_zybo.xdc** (for Zybo).

3-3-13. Click **OK**.

3-4. Run DRC.

3-4-1. Select **Tools > Report > Report DRC,** then click **OK** to run the PR-specific design rules. There should be no violations.

3-5. Run implementation.

3-5-1. Select the Design Runs tab and notice that the tools are reporting that the synthesis is out of state as we added the floorplan related constraints. Since these constraints are really needed for the implementation, we can force the synthesis run to up-to-date status.

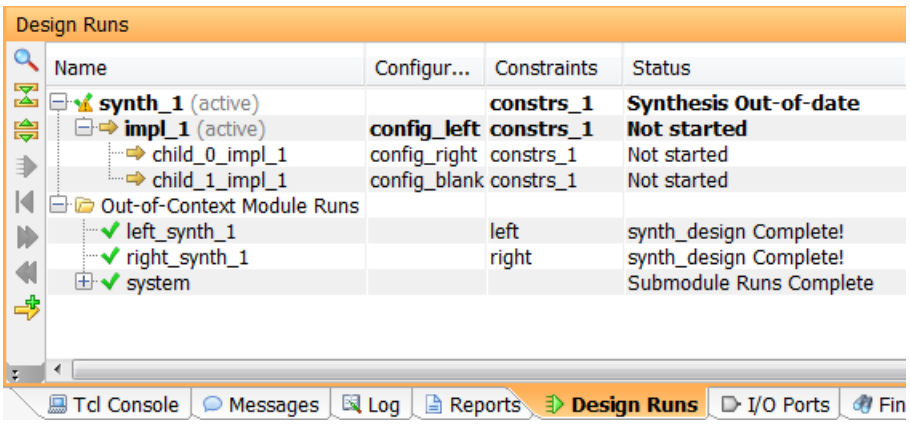


Figure 19. Design Runs

3-5-2. Right-click on the *synth_1* and select **force up-to-date**.

3-5-3. In the Flow Navigator, select **Implementation Settings**

The Project Settings window having Implementation pane selected will be displayed.

3-5-4. Scroll down and click on the browse button of the **tcl.pre** field under the *Write Bitstream* group.

3-5-5. Browse to *c:\XUP\PR\labs\icap_processor_lab*, select the **crcFrameEnable.tcl** file, and click **OK**.

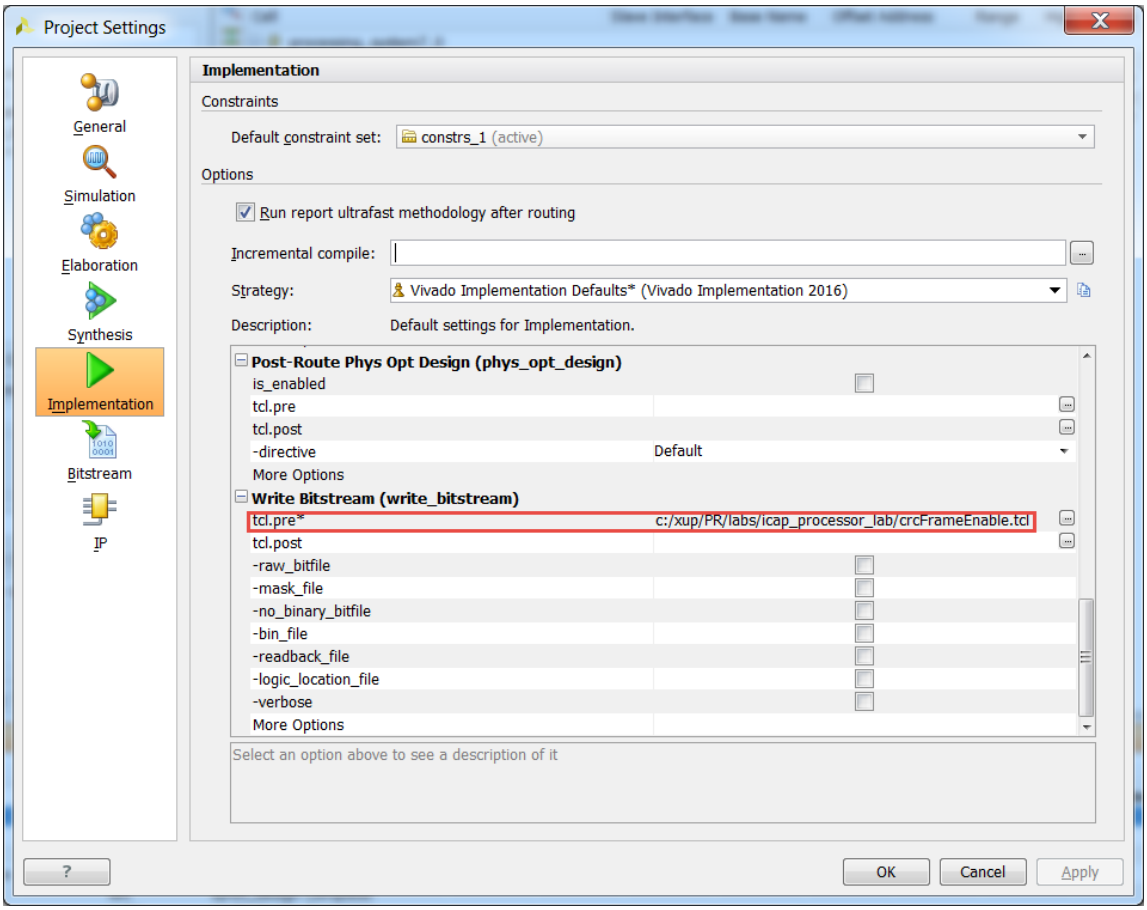


Figure 20. Setting command to enable frame-based CRC generation

3-5-6. Click **Apply** and then click **OK**.

3-5-7. In the Flow Navigator, select **Run Implementation** to run place and route on all configurations.

This action runs implementation first for *impl_1* and then for *child_0_impl_1* and *child_1_impl_1*. Behind the scenes, Vivado takes care of all the details. In addition to running place and route for the three runs with all the PR requirements in place, it does a few more tasks specific to PR. After impl_1 completes, Vivado automatically:

- Writes module-level (OOC) checkpoints for each routed *left* RM
- In a future release, these checkpoints can be used to create new configurations by mixing and matching RMs
- Carves out the logic in each RP to create a static-only design image. This is done by calling update_design –black_box for each instance
- Locks all placement and routing for this static-only design. This is done by calling lock_design –level routing
- Saves this locked static parent image to be reused for all child runs

In addition, when the child run completes, module-level checkpoints are created for the routed *right* and *blackbox* RMs. A locked static design image would be identical to the parent, so this step is not necessary.

3-5-8. When Implementation completes, select *Generate Bitstreams* and click **OK** to run the bitstreams generation.

This will create a full configuration bitstream having left shift functionality and three partial bitstreams (left, right, and greybox).

The full configuration bitstream, **top.bit** and partial bitstream **reconfig_leds_left_partial.bit** will be generated in the *impl_1* directory. The partial bitstream **reconfig_leds_right_partial.bit** will be generated in the *child_0_impl_1* directory and **reconfig_leds_greybox_partial.bit** will be generated in the *child_1_impl_1* directory.

3-5-9. Using the Windows Explorer, copy the generated full and partial bitstreams from various folders mentioned above, place them in the *c:\xup\PR\labs\icap_processor_lab\bitstreams* folder, and rename as listed below.

Source Name	New Name
reconfig_leds_greybox_partial.bit	b_led.bit
reconfig_leds_left_partial.bit	left.bit
reconfig_leds_right_partial.bit	right.bit

3-5-10. In the Tcl Console tab, make sure that the current directory is

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

3-5-11. Execute the following command to convert the partial bit files into the bin files.

```
source ./bitstream_convert.tcl
```

Generate the Software Application — Step 7

4-1. Export the hardware design and launch SDK.

4-1-1. Select **File > Export > Export Hardware…**

4-1-2. In the *Export Hardware* form, make sure that the *Include bitstream* checkbox is not checked and click **OK**.

4-1-3. Select **File > Launch SDK**

4-1-4. Click **OK** to launch SDK.

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

4-2. Create a Board Support Package enabling FAT file system.

4-2-1. In **SDK**, select **File** > **New** > **Board Support Package.**

4-2-2. Click **Finish** with the default settings (with standalone operating system). This will open the Software Platform Settings form showing the OS and libraries selections.

4-2-3. Select **xilffs** as the FAT file support is necessary to read the partial bit files.

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

4-3. Create an application.

4-3-1. Select **File** > **New** > **Application Project.**

4-3-2. Enter **TestApp** as the *Project Name*, and for *Board Support Package*, choose **Use Existing** (*standalone_bsp_0* should be the only option).

4-3-3. Click **Next**, and select *Empty Application* and click **Finish.**

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

4-3-5. Expand **General** category and double-click on **File System.**

4-3-6. Browse to *c:\xup\PR\labs\icap_processor_lab\Sources\TestApp\src* and click **OK.**

4-3-7. Select **TestApp.c** and click **Finish** to add the file to the project.

4-3-8. Right-click on TestApp and select **C/C++ Building Settings.**

4-3-9. Select **ARM v7 GCC Compiler > Symbols,** and click **+**

4-3-10. 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.

4-4. Create a zynq_fsbl application.

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

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

4-4-3. Click **Next**, select *Zynq FSBL,* and click **Finish.** This will create the first stage bootloader application called zynq_fsbl.elf

4-5. Create a Zynq boot image.

4-5-1. Select **Xilinx Tools** > **Create Zynq Boot Image.**

4-5-2. 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.

4-5-3. 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_<board>_lab\icap_processor_<board>_lab.sdk\zynq_fsbl\Debug** directory, select *zynq_fsbl.elf* and click **Open**.

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

4-5-1. 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 *top.bit* and click **Open**.

4-5-2. Click **OK**.

4-5-3. 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_<board>_lab\icap_processor_<board>_lab.sdk\TestApp\Debug** directory, select *TestApp.elf* and click **Open**.

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

4-5-5. Make sure that the output path is **c:\xup\PR\labs\icap_processor_lab** and the filename is *BOOT.bin,* and click **Create Image.**

4-5-6. Close the SDK program by selecting **File > Exit**.

4-5-7. Close the project in Vivado.

Test the Design Step 8

5-1. 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.

5-1-1. Using the Windows Explorer, copy the **BOOT.bin** from the *c:/xup/PR/icap_processor_lab/* directory on to a SD Card.

5-1-2. 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.

5-1-3. 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.

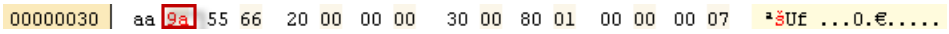


Figure 21. Corrupting the sync word

5-1-4. Similarly, open the **idcode.bin** file, change the IDCODE field as shown below, and save it.

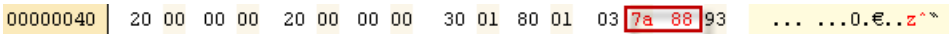


Figure 22. Corrupting the id code

5-1-5. Similarly, open the **crc.bin** file, change the content of the first frame as shown below (note the file offset [row starting at 200]), and save it. This will cause mismatch between expected and read crc.

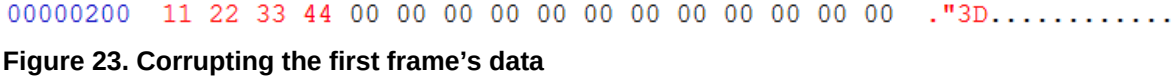


Figure 23. Corrupting the first frame's data

5-1-6. Close the Hex editor program.

5-1-7. 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.

5-2. Connect the board with one micro-USB cable to JTAG port and another micro-USB cable (ZedBoard only) to the UART port. Place the board in the SD boot mode. 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.

5-2-1. Make sure that one micro-usb cable is connected between the JTAG port of the board and the PC, and another micro-usb cable (ZedBoard only) is connected between the UART port of the board and the PC.

5-2-1. Make sure that the board is set to boot in SD card boot mode.

5-2-2. Power ON the board.

5-2-3. Start a terminal emulator program such as TeraTerm or HyperTerminal.

5-2-4. Select the appropriate COM port (you can find the correct COM number using the Control Panel).

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

5-2-6. Press **BTN7** to display a menu.

5-2-7. Follow the menu and test various reconfigurations.

5-3. Analyze waveforms using Vivado Analyzer.

5-3-1. 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.

5-3-2. Enter the following command to open the hardware manager, program the FPGA, and open the hardware manager's dashboard.

```
source run_ila.tcl
```

5-3-3. Select **xc7z020_1** (for ZedBoard) or **xc7z010_1** (for Zybo).

5-3-4. Click on the browse button of the *Probes Files* in the **properties** tab, browse to *c:\xup\PR\labs\icap_processor_lab\icap_processor_<board>_lab.runs\impl_1*, select **debug_nets.ltx** and click **OK**.

5-3-5. Select **Window > Debug Probes.**

The Debug Probes window will open up displaying the debug ports.

5-3-6. Right-click on *icap_go* and select **Add Probes to Basic Trigger Setup**.

You will see the icap_go in the Basic Trigger Setup window.

5-3-7. Change the compare value to **1**.

5-3-8. 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.

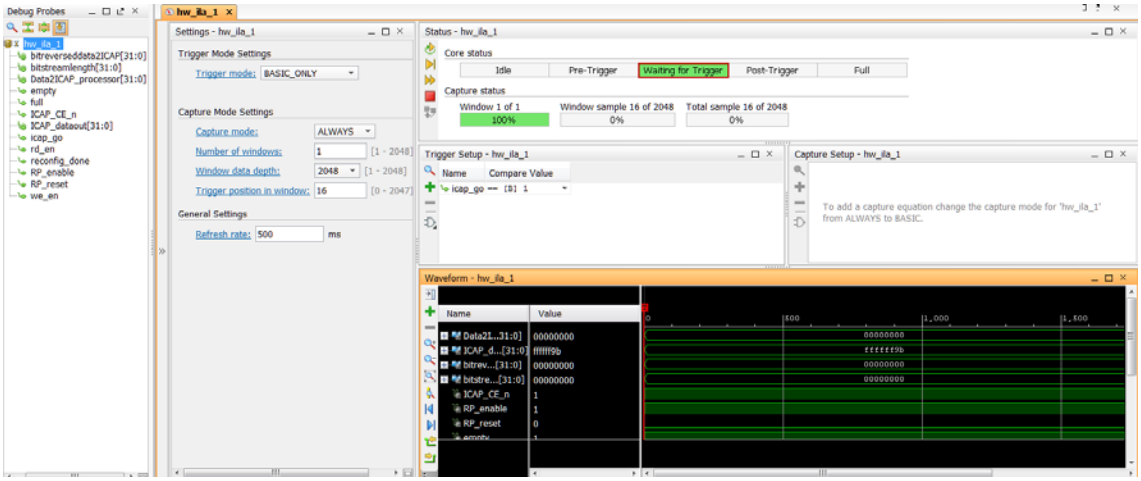


Figure 24. Waiting for the trigger condition to occur

5-3-9. In the terminal window, type **L** and observe the LEDs are shifting left and the ILA has triggered.

Zoom in into the beginning part of the waveform, click around 400 and notice the SYNC word is detected and the ICAP output changes from 0xFFFFFF9B to 0xFFFFFFDB around sample *395* indicating SYNC word is detected.

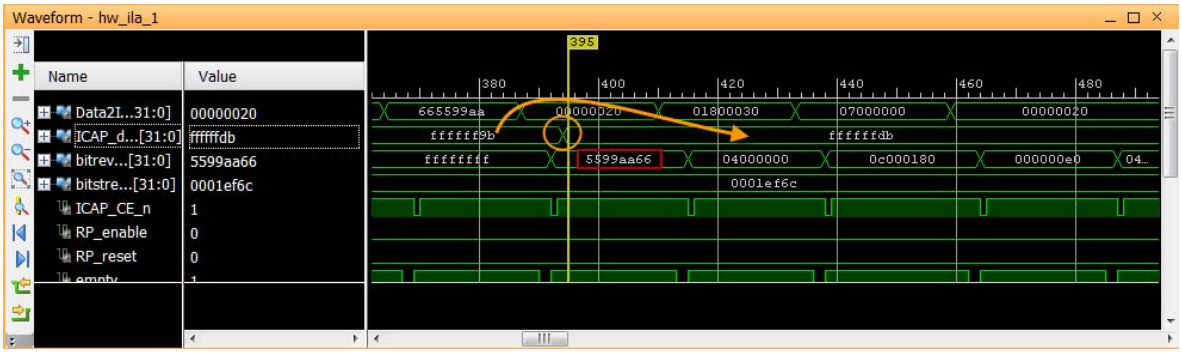


Figure 25. SYNC word detected

5-3-10. 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.

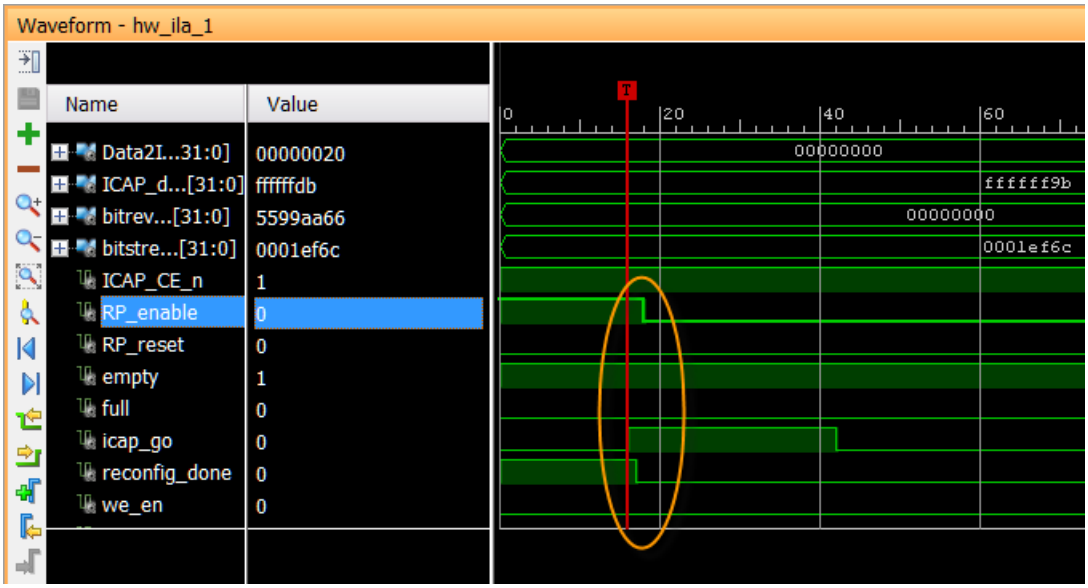


Figure 26. Activities around starting of the reconfiguration process

5-3-11. 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.

5-3-12. Type **S** to send the SYNC word corrupted bitstream and observe the waveform.

Notice that the LEDs are still shifting left, however around 390 the ICAP output did not change.

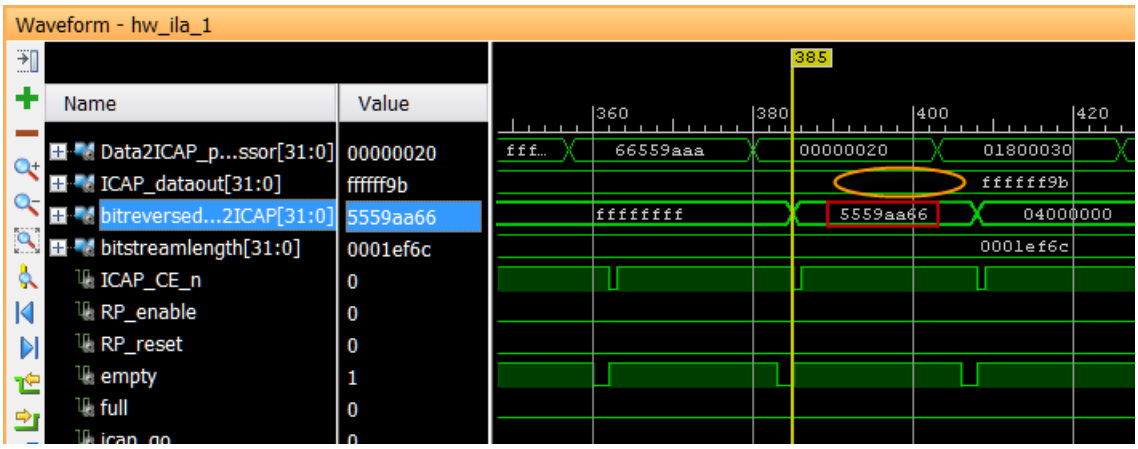


Figure 27. SYNC word error

5-3-13. Type **R** and observe the LEDs shifting right.

5-3-14. 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.

5-3-15. 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 545 the corrupted IDCODE came and around 858 the ICAP output changed to 0xFFFFFF5B followed by status change to 0xFFFFF1B indicating the reconfiguration was aborted.

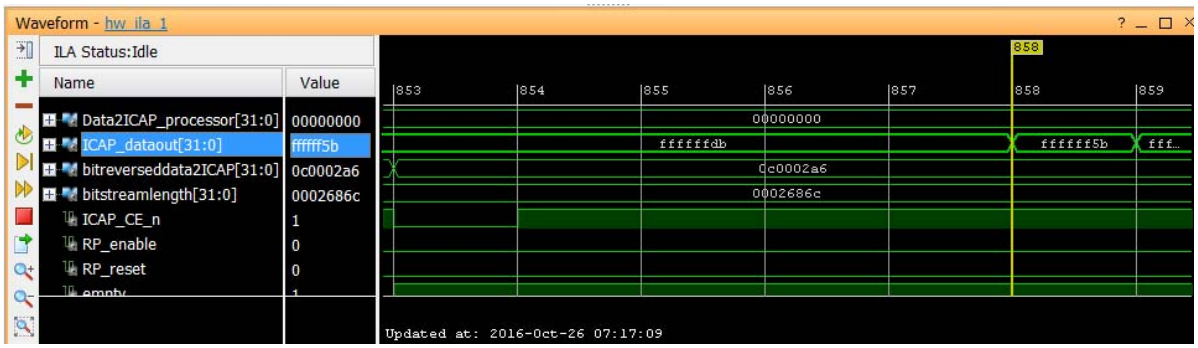


Figure 28. IDCODE word error

5-4. Use Advance Triggering to analyze end of the configuration activities

5-4-1. In the *ILA – hw_ila_1* tab, click on the drop-down button of the *Trigger mode* and select **ADVANCED_ONLY**.

5-4-2. 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). Click **OK**.

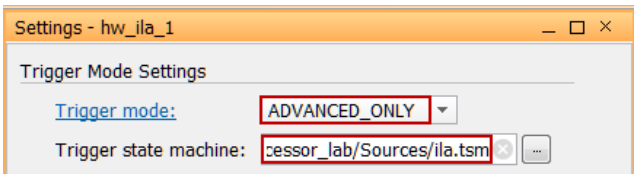


Figure 29. Setting the ILA for the advanced triggering

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

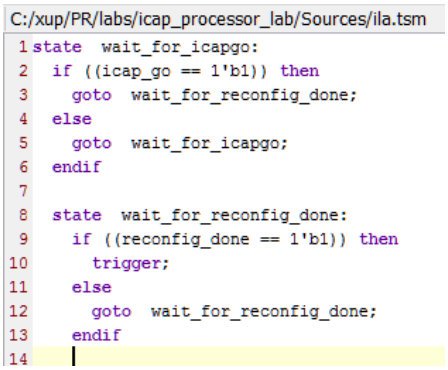


Figure 30. The ILA state machine

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.

5-4-3. Set the Trigger position to 2000 (as we are interested in what was happening before the reconfiguration is completed).

5-4-4. Click on the *Run Trigger* button to arm the ILA and waiting for user input in the terminal window.

5-4-5. 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).

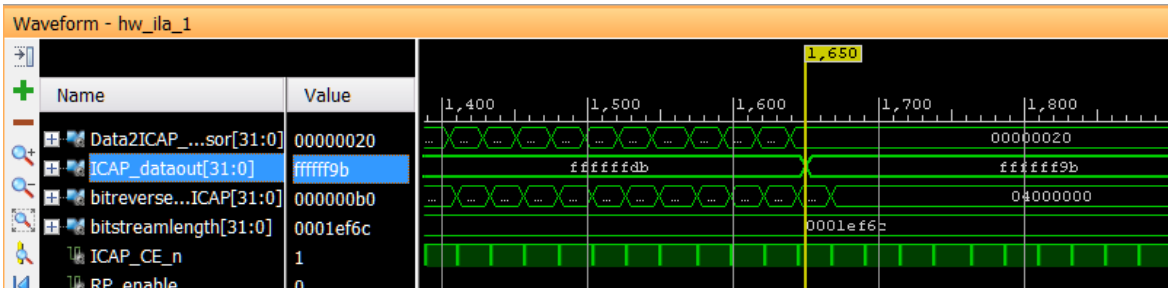


Figure 31. The triggered ILA waveform view

5-4-6. 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.

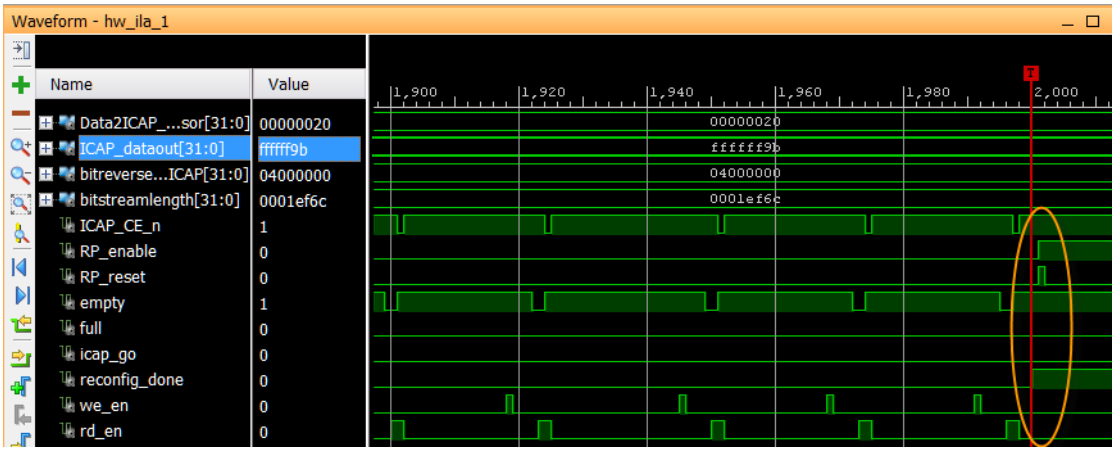


Figure 32. Zoomed view showing the activities on various signals when the reconfiguration is done

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

5-5-1. 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).

5-5-2. Set the trigger position to **2000**.

5-5-3. Run the ILA.

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

5-5-5. 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.

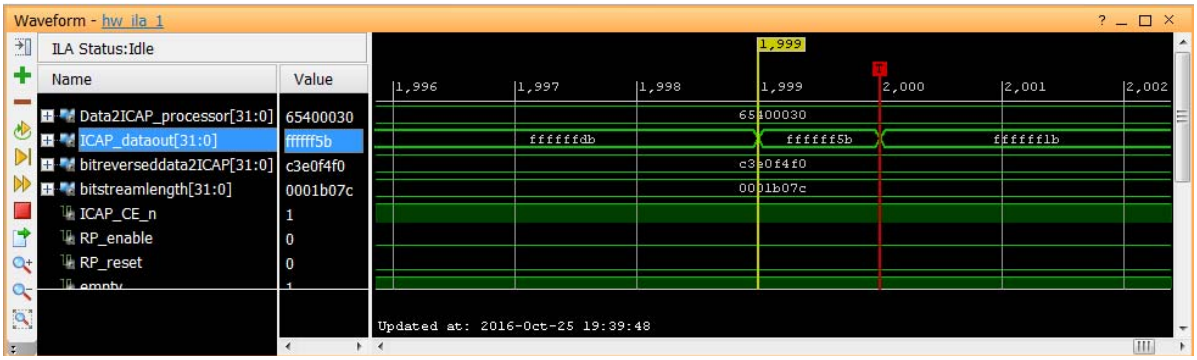


Figure 33. CRC error

5-5-6. If you scroll left (around sample 1785), you will see 0x44332211, the corrupted CRC, on the DATA2ICAP_processor bus.

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

5-5-8. 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. If you don't see anything then click on the device again.

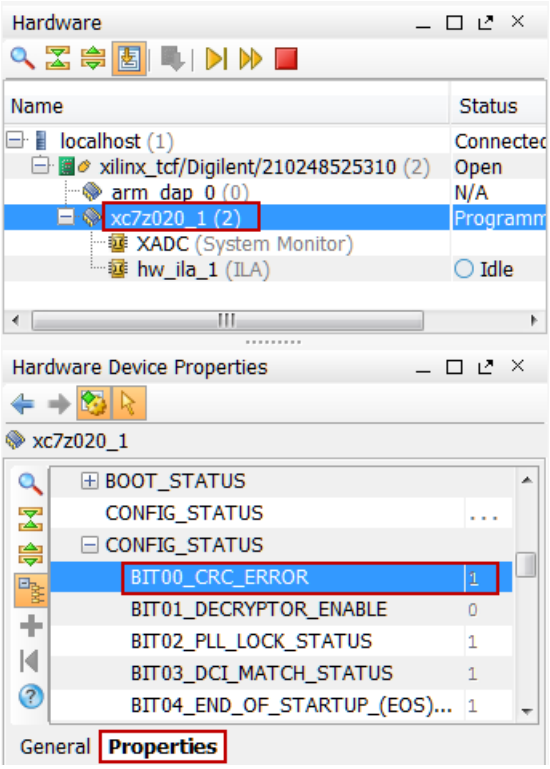


Figure 34. Verifying the CRC error in the CONFIG_STATUS register

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

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

Conclusion

This lab showed you how the project based PR flow methodology works. It also 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.