Reconfiguring User Logic Using Custom ICAP Processor and Monitoring ICAP Signals Using ChipScope Core Lab

Introduction

In this lab, you will use Xilinx Platform Studio to create a processor system. You will use Xilinx ISE to create a top-level design, which defines the MicroBlaze processor system, one Reconfigurable Partition having two Reconfigurable Modules (leftshift and rightshift), and a static control module as sub-modules. You will use PlanAhead to insert ChipScope cores to monitor internal signals for debugging purpose. You will use PlanAhead to create multiple Configurations and run the Partial Reconfiguration implementation flow to generate full and partial bitstreams. You will use the ML605 Evaluation board to verify the design in hardware using a Compact Flash memory card to initially configure the FPGA, and then partially reconfigure the device using the ICAP under user software control. You will use ChipScope Pro Analyzer to actually monitor the signals.

ICAP is accessed through a light-weight custom IP which requires bitstream length, go, and done signal to interface to a system that provides partial bitstream data. In this lab, the partial bitstream is provided by the processor system by reading the partial bitfiles from the compact flash card.

Objectives

After completing this lab, you will be able to:

- Create a processor system using Xilinx Platform Studio
- Add a processor system in ISE
- Create a PlanAhead PR project
- Insert ChipScope cores (in PlanAhead) to monitor ICAP signals
- Create various configurations for partial reconfigurations in PlanAhead
- Run the Partial Reconfiguration implementation flow
- Implement a dynamically reconfigurable design having single RP with multiple RM
- Generate partial reconfiguration bitstreams and system.ace files which can be stored on a Compact Flash memory card
- Configure an FPGA using a Compact Flash memory card and run user applications
- Monitor internal signals using ChipScope Pro Analyzer

Description

The purpose of this lab exercise is to implement a design that can be dynamically reconfigurable using a light-weight custom IP that can be used to access ICAP to write partial bitstream. Figure 1 shows a top-level system and Figure 2 shows a processor system. The design consists of a reconfigurable user logic (led shifting), having two unique capabilities (left and right shifting). User verifies functionality using HyperTerminal under user application. The dynamic modules are reconfigured using the custom IP having ICAP resource.
Figure 1. Top-Level Design

Figure 2. Processor System
The directory structure is as shown here:

```
- icap_processor_lab
  - edk
  - image
  - image_solution
- resources
  - icap_interface_v1_00_a
    - data
    - hdl
  - icap_processor_v1_00_a
    - data
    - hdl
    - netlist
  - rModule_leds
    - leftshift
    - rightshift
  - static
  - TestApp
    - src
```

The edk directory will be used for creating a processor system, and the image directory will hold the system .ace and partial bitstream files. The resources directory provides the software application. It also contains two pcores, one of which is a custom ICAP processor, as well as additional files to connect the LED controller, right and left shifting functionality, and a top-level HDL model to connect all components together.

An image_solution folder is provided with the final system.ace and partial bit files for a quick test.
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.

This lab comprises 15 primary steps: You will Create a Processor Hardware System, Create a Software Project, Create a Top-Level Design, Create a PlanAhead Project, Adding ChipScope Core, Define a Reconfigurable Partition, Add Reconfigurable Modules, Define the Reconfigurable Partition Region, Run Design Rule Checker, Create First Configuration and Implement, Create Other Configurations and Implement, Run PR_Verify, Generate Bit Files, Create Image and Test, and finally Debugging using ChipScope Analyzer.

General Flow for this Lab
Create a Processor Hardware System

Step 1

1-1.  First you will create a processor system.  This can be achieved using XPS and its BSB wizard

1-1-1.  Open XPS by selecting Start → Programs → Xilinx ISE Design Suite 13.2 → EDK → Xilinx Platform Studio.

1-1-2.  Use the default setting of Base System Builder wizard and click OK.

1-1-3.  Browse to the icap_processor_lab\edk directory and click Save.

1-1-4.  Select PLB System and then click OK.

1-1-5.  Click Next to create a new processor system.

1-1-6.  Select Xilinx in Board Vendor, Virtex 6 ML605 Evaluation Platform in Board Name, and D in Board Revision, and then click Next.

1-1-7.  Select Single-Processor System and click Next.

1-1-8.  Set the System Clock Frequency to 83.33 MHz and select 64 KB from the Local Memory drop-down menu, then click Next.

1-1-9.  Select all devices except the RS232_Uart_1, SysACE_CompactFlash, dlmb_cntlr, and ilmb_cntlr devices and click on the Remove button, as you will only use these four devices in this design.

1-1-10. Configure RS232_Uart_1 with a baudrate of 115200.

1-1-11. Click Next two more times and then click Finish.

1-1-12. A System Assembly View window will show the design.

1-1-14. Expand the clock_generator_0 instance.

1-1-15. Click on the drop-down button of CLKOUT0 port and select Make External.

1-1-16. Select Bus Interfaces tab in the System Assembly view.

1-1-17. From the IP Catalog tab, add three instances of XPS General Purpose IO (v2.00.a) from the General Purpose IO folder accepting the default settings.

1-1-18. Name the three instances as icap_go_out, icap_done_in, icap_bitstreamlength.

1-1-19. Using Windows Explorer, copy icap_interface_v1_00_a and icap_processor_v1_00_a folders from icap_processor_lab/resources folder and paste them in pcores directory under edk folder.

1-1-20. Select Project → Rescan User Repositories.
1-1-21. Add an instance of `icap_interface_v1_00_a` and `icap_processor_v1_00_a` each from the USER folder under IP Catalog tab with default settings.

1-1-22. Connect the `icap_interface_0` instance and three instances of gpio to the PLB Bus.

1-1-23. The bus connections should look like as shown in Figure 5. (You can connect two peripherals on a bus by clicking an unfilled circle or square. A peripheral is connected if the circle or rectangle is filled.)

![Figure 5. Connecting busses in System Assembly View](image)

1-1-24. Select the Addresses tab and click Generate Addresses to assign the addresses to the added instances.

1-1-25. The resulting address map should appear as shown in Figure 6.
1-1-26. Configure the icap_go_out instance to have a single bit width output, icap_done_in to have a single bit width input only, icap_bitstreamlength to have 32-bits output.

1-1-27. Select Ports tab and expand the added instances and make connections as shown in Figure 7.

1-1-28. Select the RP_enable and RP_reset ports of the icap_processor_0 instance, click on the drop-down button and select Make External.
1-1-29. Select **Hardware** → **Generate Netlist** to run PlatGen tool.

This will generate peripherals and system netlist, and system_stub.bmm files.

## Create a Software Project

### Step 2

2-1. Once the hardware system is generated, you need to create a software project, import the provided source files, compile them, and generate an executable file

2-1-1. Select **Project** → **Export Hardware Design to SDK**…

2-1-2. Uncheck **Include bitstream and BMM file** and click **Export & Launch SDK**.

2-1-3. Browse to `icap_processor_labeledk\SDK\SDK_Export` and click **OK**.

2-1-4. Click **OK** at the warning message if you get one.

2-1-5. In the Software Evaluation Kit (SDK), select **File** → **New** → Xilinx Board Support Package.

2-1-6. Notice **Standalone_bsp_0** in the **Project name** field and click **Finish** with default settings.


2-1-8. Click on the **xilfats** check box to select the **FAT** file system support for the Compact Flash card.
2-1-9. Click OK to accept the settings and close the form.

2-1-10. Select File → New → Xilinx C Project

2-1-11. Select Empty Application in the Select Project Template pane.

2-1-12. Enter TestApp as the Project Name and click Next.


2-1-14. Select TestApp in the project view, right-click, and select Import.

2-1-15. Expand General folder and double-click on File system, and browse to icap_processor_lab\resources\TestApp\src

2-1-16. Select main.c and icap_interface.h and click Finish.

2-1-17. Select TestApp in Project Explorer window, right-click and select Generate Linker Script.

2-1-18. Change the Heap and Stack sizes to 2048 (2 KB) each, click Generate and click Yes.
Create a Top-Level Design

Step 3

3-1. You will create a top-level design for the lab

3-1-1. Open ISE by selecting Start → Programs → Xilinx ISE Design Suite 13.2 → ISE Design Tools → Project Navigator.

3-1-2. Select File → New Project... to create a new ISE project.

3-1-3. Browse to the set the Location to the icap_processor_lab directory, and then enter top as the project Name (note that the directory path is also appended with the name)

3-1-4. Click Next.

3-1-5. Select Virtex6 as the family, XC6VLX240T as the Device, FF1156 as the Package, -1 as the Speed, and VHDL as the Preferred Language. Click Next, and click Finish.

3-1-6. In the Design Hierarchy window, right-click on the device entry and select Add Copy of Source as you will be using a previously created VHDL model.

3-1-7. Browse to the icap_processor_lab\resources folder, and select top.vhd and click Open.
3-1-8.  Click OK.

3-1-9.  Right-click on the device entry again, select Add Source (not copy of source), browse to the icap_processor_lab\edk folder, select system.xmp, and click Open.

3-1-10. Click OK and click Yes to reload the report data in XPS.

3-1-11. Select the top module in the Sources window, right-click on the Synthesize - XST process in the Processes window, and select Process Properties.

3-1-12. In the GUI, change keep_heirarchy switch setting from No to Yes and click OK to close the form.

3-1-13. Select the top module in the Sources window and double-click on the Synthesize - XST process in the Processes window.

This will call platgen in the background to synthesize the processor system and then synthesize the top-level design using XST. Note that the led_control and rModule_leds modules are set as black box.

3-1-14. Close Project Navigator.
Create a PlanAhead Project  Step 4

4-1. Now that you have generated all the netlist files required for the design, you will use PlanAhead to floorplan the design, define Reconfigurable Partitions, add Reconfigurable Modules, run the implementation tools, and generate full and partial bitstreams. In this step you will create a PlanAhead project.

4-1-1. Open PlanAhead by selecting Start → Programs → Xilinx ISE Design Suite 13.2 → PlanAhead → PlanAhead.

4-1-2. Click on Create New Project.

4-1-3. Click Next, then use the browse button to select the icap_processor_lab directory for the Project location.

4-1-4. Click Next.

4-1-5. Select the Specify synthesized (EDIF or NGC) netlist option, check Set PR Project option and click Next.
4-1-6. Click on the browse button and select top.ngc file from icap_processor_lab\top

4-1-7. Click Add Directories… button, browse to icap_processor_lab\resources\static folder and click Select. Similarly, add the implementation directory under icap_processor_lab\edk

4-1-8. Click Next.

4-1-9. In Add Constraints (optional), select system.ncf entry and click (X) button to remove the file.

4-1-10. Click on the Add Files… button browse to icap_processor_lab\resources and select the top.ucf file and click OK.

4-1-11. Click Next.
4-1-12. Make sure that xc6vlx240tff1156-1 device is selected in **Default Part** form.

4-1-13. Click **Next** and then click **Finish**.

4-1-14. Click on **Netlist Design** ( ) in the Project Manager window (left) to load the netlists for the design.

A warning message indicates that one instance will be converted to a black box, as the netlist file for it is missing. This is expected, as no netlist has been associated with this module yet. Click **OK** to continue.

4-1-15. The Netlist pane will show the modules present in the design.

![Netlist pane](image)

**Figure 14. Design hierarchy in PlanAhead**
Adding ChipScope Core

Step 5

5-1. You can add ChipScope core in two ways: (i) use ChipScope inserter and (ii) set up in PlanAhead. We will use the second method

5-1-1. Select Tools → Set up ChipScope...

5-1-2. Click Next.

5-1-3. Click on the Add/Remove Nets ... button.

5-1-4. In the Find Net Criteria pane, enter the signal name that you want to monitor (e.g. *icap_processor_0/*).

5-1-5. Select the desired signals in the left pane and add them to the right side window. Make sure that the signals shown in the figure below are added.

Figure 15. Signals to be monitored using Chipscope

5-1-6. When done adding the signals, click OK.

5-1-7. At this stage, you can select which signal needs to be TRIG, DATA, or both and what clock domain it belongs to (if you have multiple clock domains).
Figure 16. Added signals

5-1-8. Right click on any signal and select one of the options for that signal (a). Make sure that the signals are set as TRIG only, DATA only, and TRIG and DATA as shown in (b) below. Only icap_go, ICAP_CE, bitstream_done, and reconfig_done need to be set as TRIG.
Figure 17. Selecting signals to be monitored and/or used as trigger

5-1-9. Click Next and then Finish.

Define a Reconfigurable Partition

6-1. In this design you have one Reconfigurable Partition and it has to be defined explicitly

6-1-1. Select the reconfig_leds instance in the Netlist window, right-click on it, and select Set Partition...

6-1-2. Click Yes to save the project.

6-1-3. Click Next twice.

6-1-4. Enter led_BB as the RM name since the partition does not have a defined netlist yet. Select the Add this Reconfigurable Module as a black box... option and click Next.
Figure 18. Creating a Black Box Reconfigurable Module

6-1-5. Click Finish.

Note that the black box icon has changed from I to a diamond shape.

Add Reconfigurable Modules

Step 7

7-1. In this design you have two Reconfigurable Modules for the Reconfigurable Partition that you will add next

7-1-1. Select the reconfig_leds instance in the Netlist window, right-click on it, and select Add Reconfigurable Module…

7-1-2. Click Next.

7-1-3. Enter left in the name field and click Next, leaving Netlist already available for this Reconfigurable Module selected.
Figure 19. Adding the left shifting Reconfigurable Module

7-1-4. Browse to \icap_processor_lab\resources\rModule_leds\leftshift and select rModule_leds.ngc.

Figure 20. Locating the left version of rModule_leds.ngc

7-1-5. Click Next twice and then Finish. The Netlist pane will show the added RM entry – expand rModule_leds to view.

7-1-6. Follow the same steps to add the right RM from \icap_processor_lab\rModule_leds\rightshift directory selecting rModule_leds.ngc from that directory, naming the RM as right.

The Netlist pane should now should three Reconfigurable Modules (including the black box) for the leds Reconfigurable Partition. Note that the right module is active (with a check mark) as it was the most recent netlist to be added to the project.
Define the Reconfigurable Partition Region

Step 8

8-1. Next you must floorplan the RP region. Depending on the type and amount of resources used by each RM, the RP region must be appropriately defined so it can accommodate any RM variant

8-1-1. Select `pblock_reconfig_leds` instance in Physical Constraints window, right-click, and select Set Pblock Size.

Note: select Window → Physical Constraints if the Physical Constraints cannot be found.

8-1-2. Zoom to the top half of the FPGA. Move the cursor in the floorplan window and draw a box that bounds `SLICE_X26Y160:SLICE_X31Y164`

Note that the current grid coordinates are reported on the bottom toolbar in the PlanAhead window.

8-1-3. Click OK to include SLICE as the resources to be reconfigured.
Run Design Rule Checker  

**Step 9**

9-1. **It is always good idea to run a design rule checker so you can catch errors as soon as possible**

9-1-1. Select **Tools → Run DRC.**

9-1-2. Deselect **All Rules**, select **Partial Reconfig** and **ChipScope**, and then click **OK** to run the PR-specific design rules.
Figure 24. Running Design Rule Checks

You will see just warnings indicating that RMs have not been implemented.

Create First Configuration Step 10

10-1. Now you can create and implement the first Configuration

10-1-1. Select Tools → Options…

10-1-2. Select Strategies in the left pane, click drop-down button, and select ISE 13.

10-1-3. Click on the “+” button to create a new strategy.
10-1-4. Name it ISE13_BM and click OK.

10-1-5. Type `-bm ..\..\\edk\implementation\system_stub.bmm` in the More Options under the Translate (ngdbuild) process.

10-1-6. Click Apply and then click OK.

10-1-7. Select config_1 in the Design Runs tab.

10-1-8. Select General tab in the Implementation Run Properties and change config_1 to config_right.

10-1-9. Click Apply.

10-1-10. Select Options tab, select ISE13_BM (ISE 13) in the Strategy field, and click Apply.

   Make sure that the -bm path is pointing to correct file and directory, if not then change it.

10-1-11. Select Partitions tab, click on drop-down button under the Module Variant column, and select right as the variant.

10-1-12. Click Apply.

10-1-13. Select config_right in the Design Runs tab, right-click and select Launch Runs...

10-1-14. Click OK to run the implementation and Save to save the project.
When the implementation run completes, a dialog box with four choices is presented. Select Promote Partitions.

10-1-15. Click OK twice to promote the partitions.

Create Other Configurations

11-1. After you have created the first configuration, the static logic implementation will be reused for the rest of the configurations. That is why you have promoted the implemented configuration. Next you will create the desired number of additional configurations and implement them.

11-1-1. Close the project and re-open the project.

This step is required for 13.2 version in order for the tools to see that the implemented config_right configuration is promoted. Ignoring this step will generate error when new configuration runs are created and implementation is run.

11-1-2. Click on Netlist Design.

11-1-3. Select Flow → Create New Runs…

11-1-4. Click Next twice.

11-1-5. A Create New Runs window will appear with run Name as config_1.

Make sure that the ISE13_BM is selected as the strategy.

11-1-6. Rename config_1 to config_left.

11-1-7. Click More to create another configuration named config_1.

11-1-8. Rename config_1 to config_bb.

11-1-9. Click Next.

11-1-10. Select Do not launch now, click Next and then click Finish.

11-1-12. In the corresponding Implementation Run Properties window select Partitions tab, and then using drop-down button select left module variant.

11-1-13. Click Apply.

11-1-14. Similarly, for the config_bb run, select led_BB module variant and click Apply.

11-1-15. Select config_bb and config_left in the Design Runs window, right-click and select Launch Runs…

11-1-16. Click OK to run the implementation.

11-1-17. Click Cancel after run is completed.

Run PR_Verify  

Step 12

12-1. You must ensure that the static implementation, including interfaces to reconfigurable regions, is consistent across all Configurations. To verify this, you can run the PR_Verify utility

12-1-1. Select any of the Configurations in the Configurations window, right-click, and select Verify Configuration …

Figure 27. Verify all Configurations

12-1-2. Shift-click to select all Configurations and click OK.

12-1-3. The PR_Verify utility will run and will report that there were no errors.
Generate Bit Files  

**Step 13**

13-1. **After all the Configurations have been validated by PR_Verify, full and partial bit files must be generated for the entire project.** In order to create corrupted bit files from the generated partial bit files, it is important to change the default Bitgen’s properties. By default, if there is any error in the configuration file, the FPGA will reset to 0 and since we are running off the compact flash, the FPGA will not reconfigure and the system will hang. In order to prevent this while testing with corrupted IDCODE, it important to add `-g configfallback:Disable` as an option in the more options field of bitgen.

13-1-1. Use a shift-click to select all three design runs (config_right, config_left, and config_bb) in the Design Runs window, right-click, and select Generate Bitstream.

13-1-2. The properties form will be displayed, add the following option to change the default value and then click OK.

   `-g configfallback:Disable` in the more options field.

13-1-3. This will run the bitstream generation process and generate full and partial bitstreams.

   The bit files will be placed in the `config_left`, `config_right`, and `config_bb` directories under the `icap_processor_lab\project_1\project_1.runs\` directory.

13-1-4. Save the project and close PlanAhead.

Create Image and Test  

**Step 14**

14-1. **You will create images, download them to the board and do the test.**

14-1-1. Open a shell by selecting Project → Launch Xilinx Shell from XPS or Start → Programs → Xilinx ISE Design Suite 13.2 → Accessories → ISE Design Suite Command Prompt.

14-1-2. In the shell, change directory to `icap_processor_lab\image` and then execute the following command to generate download.bit file (having software component included) from impl_left.bit (having just hardware component)

   ```sh
data2mem -bm ../edk/implementation/system_stub_bd -
bt ../project_1/project_1.runs/config_left/config_left.bit -
bout bd ../edk/SDK/SDK_Export/TestApp/Debug/TestApp.elf tag microblaze_0 -o
b download.bit
```
Hint: Copy the command text from this document and paste in the Bash shell by right clicking on the title bar and selecting **Edit → Paste**.

This will generate the download.bit in the `image` directory.

**14-1-3.** In the shell, execute following command to generate `system.ace` in the `image` directory.

```
xmd -tcl genace.tcl -jprog -target mdm -hw download.bit -board ML605 -ace system.ace
```

**14-1-4.** Using Windows Explorer, copy `icap_processor_lab/project_1/project_1.runs/config_right/config_right_reconfig_leds_right_partial.bit` into the `image` directory; rename it as `right.bit`.

**14-1-5.** Similarly, using Windows Explorer, copy `icap_processor_lab/project_1/project_1.runs/config_left/config_left_reconfig_leds_left_partial.bit` into the `image` directory; rename it as `left.bit`.

**14-1-6.** Similarly, using Windows Explorer, copy `icap_processor_lab/project_1/project_1.runs/config_bb/config_bb_reconfig_leds_led_bb_partial.bit` into the `image` directory; rename it as `blank.bit`.

**14-1-7.** Using a Hex Editor, open `left.bit` file and make change to the SYNC word so it looks like as shown below and then save the file as `sync.bit`.

```
000000a0 22 00 44 ff ff ff ff ff ff ff ff ff aa 9a 55 66 20
```

**14-1-8.** Similarly, open the `left.bit` file again, change the IDCODE field as shown below, and save it as `idcode.bit`.

```
000000c0 00 00 30 01 80 01 04 45 00 93 30 00 80 01 00
```

**14-1-9.** Similarly, open the `left.bit` file again, change the CRC field as shown below, and save it as `crc.bit`.

```
000000ca 00 00 01 49 ee ef 15 30 00 80 01 00 00 00 0d 20
```

**14-1-10.** Close the Hex editor program.

**14-1-11.** Place a blank Compact Flash memory card in a CompactFlash writer.

**14-1-12.** Using Windows Explorer, copy the six partial bit files and system.ace from `icap_processor_lab/image` folder to the CompactFlash card.

**14-1-13.** Place the CompactFlash card in the ML605 board, and set the mode pins (SW3) to 00010101 to configure the FPGA from the CompactFlash. Connect your PC to the ML605 with an RS-232 cable. Start a HyperTerminal window, connecting via COM1 at 115200 baud.
Debugging using ChipScope Analyzer

15-1. You will debug the design using ChipScope Analyzer

15-1-1. Connect JTAG programming cable to the board.

15-1-2. Power ON the ML605 board.


15-1-4. In the Analyzer window, select JTAG Chain → Xilinx Platform USB Cable… and click OK.

15-1-5. Click OK when the devices in the JTAG Chain are displayed.

15-1-6. Select File → Import and browse to C:\xup\PartialReconfiguration\labs\icap_processor_lab\project_1\project_1.runs\config_left directory and select debug_nets.cdc. Click Open and then click OK.

15-1-7. In the trigger window, set the trigger condition of ICAP_GO to 1 as the signal is asserted when it is high.

15-1-8. In the capture window, enter 32 as the position so we can see 32 samples history.

![Figure 28. Trigger window setup](image)


15-1-10. Rename /system_i/icap_processor_0/icap_processor_0/icap_inst/F1/BU2/dout to ICAP_Datain.

15-1-11. Select ICAP_Datain, right-click and select Add to View → Listing.
15-1-12. In the signals window on the left, rename
/system_i/icap_processor_0/icap_processor_0/icap_inst/ICT/ICAP_dataout to ICAP_Dataout
and add it to the listing window.

15-1-13. Rename /system_i/icap_processor_0/icap_processor_0/icap_inst/ICT/I1/F1/ICAP_CE to
ICAP_CE and add it the listing window.

15-1-14. Similarly, rename rest of the signals, removing the preceding path.

15-1-15. Click on Run button and observe that it is waiting for the trigger condition to occur.

15-1-16. Go to the HyperTerminal window and type L or R to select to load the partial bitstream and
trigger the analyzer.

15-1-17. The waveform window will look like

![Waveform window showing captured signals](image)

Figure 29. Waveform window showing captured signals

15-1-18. Note that the RP_enable and reconfig_done are negated as the reconfiguration has started.
The ICAP_CE is asserted (low) for several clock cycles as the 127 words of bitstream data were
available in the FIFO. You will also see that at sample 168, the ICAP_CE goes low again as the
software puts another word. More data are being written, but it takes longer so only initial data
are captured.

15-1-19. In the capture window, enter 0 as the position.

15-1-20. Click on Run button and observe that it is waiting for the trigger condition to occur.

15-1-21. Go to the HyperTerminal window and type L or R to select to load the partial bitstream and
trigger the analyzer.

15-1-22. The listing window close to the trigger condition will look like.
Figure 30. Listing window

15-1-23. Zoom in into the waveform windows and study the data going to ICAP.

15-1-24. Now change the trigger condition to set Bitstream\_done to 1 and icap\_go to “x”. Set the trigger position to 900 as we want to see what was being send towards the end of the bitstream file.

15-1-25. Click on Run button and observe that it is waiting for the trigger condition to occur.

15-1-26. Go to the HyperTerminal window and type L or R to select to load the partial bitstream and trigger the analyzer.
Figure 31. Trigger conditions and waveform output for good bitstream

15-1-27. Notice that when the bitstream_done goes high, reconfig_done is asserted. The RP is enabled and is also reset for one clock cycle. The ICAP_Dataout changes from 0xDF to 0x9F when it receives the DI-SYNC command.

15-1-28. Now change the trigger condition to detect when ICAP_CE to 0, bitstream_done to “x”, and trigger position to 32, and arm the analyzer (click on RUN button)

15-1-29. In the HyperTerminal window type s to send the SYNC word corrupted data.
Figure 32. Trigger condition and waveform windows for SYNC corrupted file

15-1-30. Notice that the ICAP_dataout has not changed from 9F to DF as the SYNC word was not detected.

15-1-31. Now change the trigger condition to detect IDCODE word error, ICAP_CE to X, bitstream_done to 1, icap_go to X, and trigger position to 900, and arm the analyzer (click on RUN button)

15-1-32. In the HyperTerminal window, type I to load the IDCODE corrupted file. Observe that when the bitstream_done is asserted, the ICAP_dataout shows 1F value indicating bit 7=0, meaning configuration error has occurred. The system did not stop as we had used the switch to not to disable it when configuration error takes place while generating bitstreams.

Figure 33. Waveform window for IDCODE corrupted file (end of configuration)

15-1-33. If you change the trigger condition to ICAP_CE=0, bitstream_done=X, and trigger position to 32, and then arm the analyzer, you will see that the ICAP_Dataout changes from 5F => DF => 1F when the IDCODE check command is issued indicating IDCODE error. This error remains set until the end of the bitstream (as seen in the previous figure)
15-1-34. Now change the trigger condition to detect CRC error. Set ICAP_CE=X, bitstream_done=1, and trigger position to 900, and then arm the analyzer, you will see that the ICAP_Dataout changes from DF => 1F when the CRC check command is issued indicating CRC error. This error remains set until the end of the bitstream.

15-1-35. When done, close the ChipScope Analyzer.

Conclusion

This lab showed you steps involved in creating a processor system using XPS, generating one RP and two RMs with simple led shifting functionality, synthesizing peripherals, and creating top-level design in ISE. It also showed you how ChipScope Logic Analyzer core can be inserted in PlanAhead, and how ChipScope Analyzer can be used to monitor the signals and subsequently debug a design. Full bitstream as well as partial reconfiguration bitstreams were generated using PlanAhead. You verified the functionality using the ML605 board.