

Reconfiguring with HW-SW Triggers Using PRC Lab

Introduction

In this lab, you will use the Partial Reconfiguration Controller (PRC) core to reconfigure a design that has two Reconfigurable Partitions (RP), each having two Reconfigurable Module (RM). You will go through the PR flow to generate the full and partial bitstreams which will be used to perform reconfiguration upon hardware events and software commands.

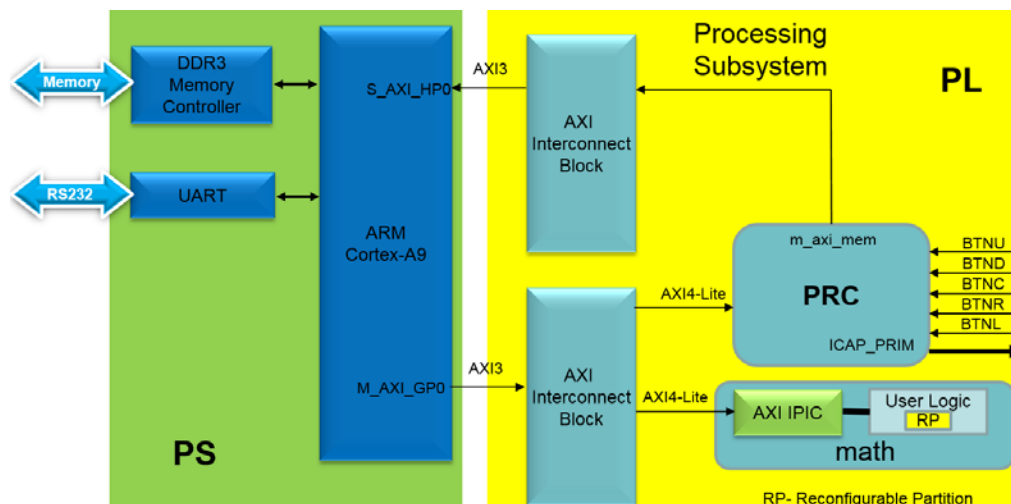
Objectives

After completing this lab, you will be able to:

- Use a Tcl script to generate a Vivado IPI design having a PS7 sub-system and the PRC along with the math RP
- Configure the PRC for both software and hardware triggers
- Use various Tcl scripts to synthesize the RMs, floorplan the design, add the RMs, create multiple configurations, implement the design and generate the full and partial bitstreams for various configurations
- Use Xilinx SDK program to create an application and a bootable BOOT.bin file
- Copy the generated bitstreams and the BOOT.bin on a SD Card and verify partial reconfigurable design functionality

Design Description

The purpose of this lab exercise is to implement a design that is dynamically reconfigurable using the PRC. The design, shown in Figure 1, consists of the PRC and two RPs. Each RP has two RMs. The two RPs are: math and led. The math RP consists of two functions: addition and multiplication, whereas the led RP consist of right and left shifting pattern of LEDs. User interacts with math RP using a terminal emulator program whereas interaction with led RP is achieved using push-buttons. The dynamic partial reconfigurable modules are updated either through menu using the PRC's software triggers capability or through push-buttons using the hardware triggers. **Since Zybo does not have five push-buttons on the PL side, we will use one of the two GPIO buttons (BTN4) through EMIO in the Zybo design.**



Hardware Triggers

Top

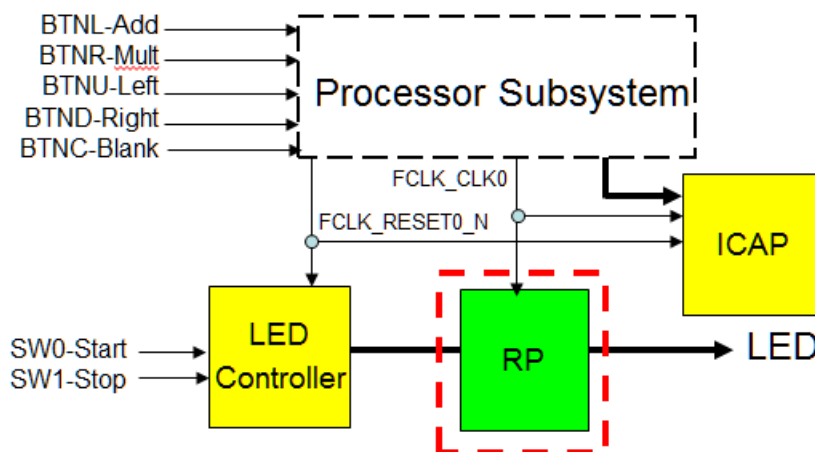
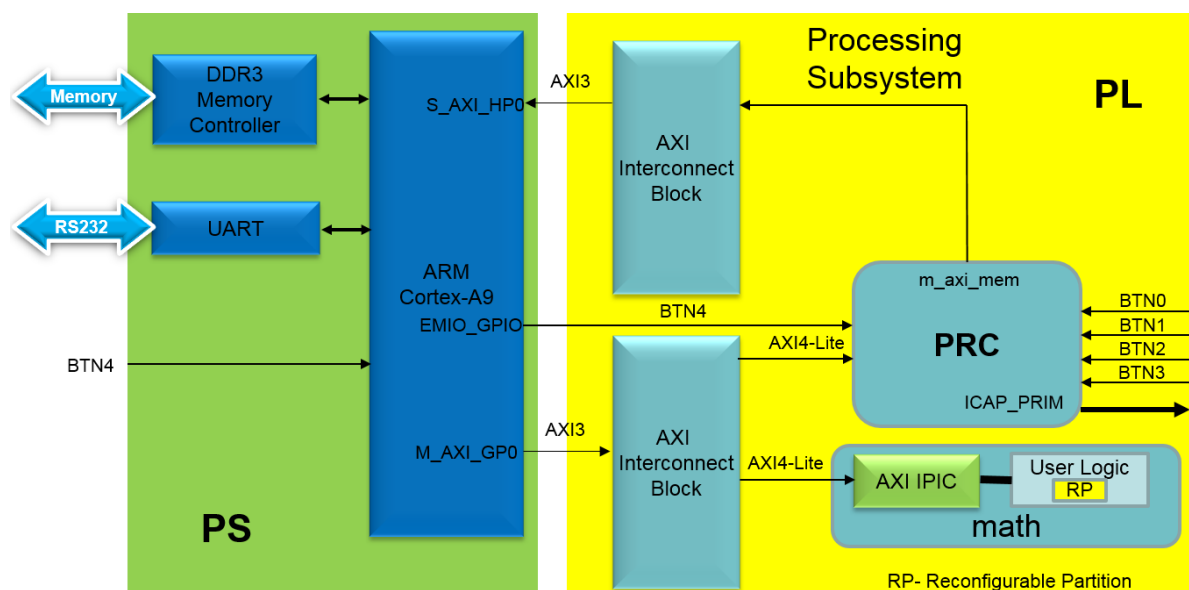


Figure 1. A Complete System (ZedBoard)



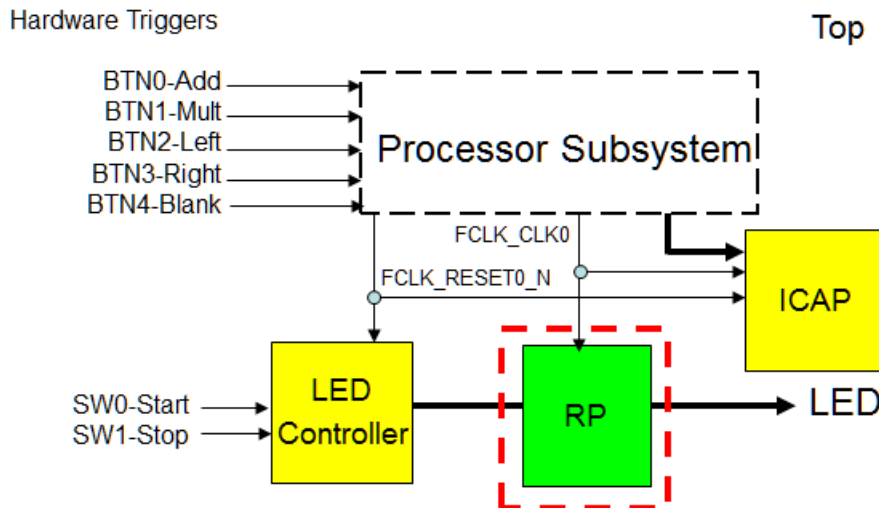
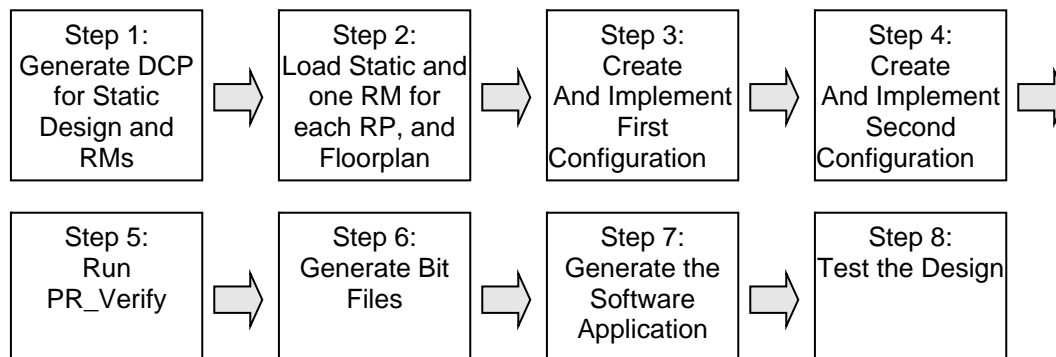


Figure 1. A Complete System (Zybo)

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 2016.3 program and execute the provided Tcl script to create the design check point for the static design having two RPs.

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

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

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

- Set the project settings to point to the provided ip repository (having math ip)
- Instantiate ZYNQ PS with SD 0 and UART 1 peripherals enabled, M_GP0 and S_HP0 (in 32-bit mode) enabled, and FCLK_CLK0 and FCLK_RESET0_N ports enabled. **Zybo only:** Enable PS7 GPIO and EMIO (1 bit width) to use BTN4
- Add the math ip and run connection wizard to connect it to the M_GP0 interface instantiating axi_interconnect
- Add the prc instance, add another axi_interconnect instance. Connect the added axi_interconnect instance to the S_HP0 interface on one side and prc's m_axi_mem interface on the other side
- Configure the prc instance to provide the AXI Lite interface so the software can read and write various registers, change the Specify the CAP arbitration protocol option to 1) Latency has not been added to arbiter signals, specify the number of clock domain crossing stages to 2 (since we have a single clock domain design) (Figure 2)

The screenshot shows the 'Global Options' tab of a configuration window. The 'Virtual Socket Manager Options' sub-tab is active. The 'Enable the AXI Lite interface' checkbox is checked. Below it, 'Set the polarity of reset and icap_reset' is set to '0'. 'Managed device type' is 'The PRC will manage Virtual Sockets on a 7 Series device'. 'Specify the CAP arbitration protocol' is set to '1) Latency has not been added to arbiter signals'. 'Specify the FIFO depth' is '32'. 'Specify how the FIFO should be implemented' has two radio buttons: 'Block RAM' (unselected) and 'Distributed RAM' (selected). At the bottom, 'Specify the number of Clock domain crossing stages' is set to '2'.

Figure 2. Selecting AXI Lite interface and clock domain crossing stages

- Support two RPs (Figure 3, 4, and 5) each having 3 RMS (including blanking RM). The PRC is designed to have power of 2 number of RMS so even when 3 is selected 4 RMS support will be included. The Virtual Socket Manager needs to be created first; enter meaningful name in the Enter a new here field which will appear in the Virtual Socket Manager to configure drop-down field. Enter the number of RMS in that RP. Next create enter a meaningful name in the Enter a new name here field under the Reconfigurable Module Options, the name will appear in the Reconfigurable Module to configure drop-down field. Enter known bitstream address and size, if any, otherwise leave them 0 (Figure 3). The bitstream address and size can be a known value or it can be anything if the application will be writing the correct values to the corresponding registers during the run-time

Virtual Socket Manager Options		Reconfigurable Module Options	
Virtual Socket Manager to configure	rp math	Reconfigurable Module to configure	add
Virtual Socket Name	rp_math ID: 0	Reconfigurable Module Name	add
<input type="checkbox"/> Has Status Channel	<input type="checkbox"/> Has Control Channel	Shutdown type	Not Required
<input type="checkbox"/> Start in Shutdown	<input checked="" type="checkbox"/> Shutdown on error	Startup type	Not Required
<input type="checkbox"/> Skip RM startup after reset		Reset type	Not Required
<input type="checkbox"/> Has PoR RM	add	Duration of Reset	1
Number of RMs allocated	4 [2 - 128]	Bitstream 0 address	0x200000
		Bitstream 0 size (bytes)	0

Figure 3. Creating Virtual Socket Manager for math RP, defining number of RMs, adding new RM called add, and assigned bitstream address and size

- Create another RM by clicking on the New Reconfigurable Module button and then enter the relevant information (Figure 4). and create a blanking RM (Figure 5). Then assign the trigger IDs to the RMs (Figure 6)

Reconfigurable Module Options	
Reconfigurable Module to configure	mult
Reconfigurable Module Name	mult
Shutdown type	Not Required
Startup type	Not Required
Reset type	Not Required
Duration of Reset	1
Bitstream 0 address	0x00300000
Bitstream 0 size (bytes)	0

Figure 4. Creating another RM

- Create another RM by clicking on the New Reconfigurable Module button and then enter the relevant information (Figure 5). Then assign the trigger IDs to the RMs (Figure 6)

Reconfigurable Module Options	
Reconfigurable Module to configure	b math
Reconfigurable Module Name	b_math
Shutdown type	Not Required
Startup type	Not Required
Reset type	Not Required
Duration of Reset	1
Bitstream 0 address	0x00600000
Bitstream 0 size (bytes)	0

Figure 5. Creating blanking RM

Trigger Options	
Number of Hardware Triggers	3
Number of Triggers allocated	4
First trigger to display	0
Trigger ID	Reconfigurable Module to Load
0	add
1	mult
2	b math
3	add

Figure 6. Setting up trigger IDs

- Next, create the second RP by clicking on the New Virtual Socket Manager button, entering `rp_shift` in the name field and entering 4 in the Number of RMs allocated field (Figure 7)

New Virtual Socket Manager **New Reconfigurable Module**

Virtual Socket Manager Options

Virtual Socket Manager to configure: `rp shift`

Name (ID): `rp_shift (1)`

Enter a new name here:

☐ Has Status Channel ☐ Has Control Channel

☐ Start in Shutdown ☒ Shutdown on error

☐ Skip RM startup after reset

☐ Has PoR RM

Number of RMs allocated: [2 - 128]

Figure 7. Creating and setting up the second RP

- Next, create the RMs for the second RP by clicking on the New Reconfigurable Module button, entering left in the name field and entering 4 in the Number of RMs allocated field (Figure 8)

New Reconfigurable Module

Reconfigurable Module Options

Reconfigurable Module to configure: left

Name (ID): left (0)

Enter a new name here:

Shutdown type: Not Required

Startup type: Not Required

Reset type: Not Required

Duration of Reset: 1 [1 - 256]

Bitstream 0 address: 0x00400000

Bitstream 0 size (bytes): 0

Figure 8. Creating first RM for the second RP

- Similarly create two more RMs, right and b_led for the second (Figure 9)

Reconfigurable Module Options

Reconfigurable Module to configure: right

Name (ID): right (1)

Enter a new name here:

Shutdown type: Not Required

Startup type: Not Required

Reset type: Not Required

Duration of Reset: 1

Bitstream 0 address: 0x00500000

Bitstream 0 size (bytes): 0

Reconfigurable Module Options

Reconfigurable Module to configure: b_led

Name (ID): b_led (2)

Enter a new name here:

Shutdown type: Not Required

Startup type: Not Required

Reset type: Not Required

Duration of Reset: 1

Bitstream 0 address: 0x00700000

Bitstream 0 size (bytes): 0

Figure 9. Creating two more RMs for the 2nd RP

- Then assign the trigger IDs to the RMs (Figure 10)

Trigger Options

Number of Hardware Triggers: 3 [0 - 512]

Number of Triggers allocated: 4 [2 - 512]

First trigger to display: 0

Trigger ID	Reconfigurable Module to Load
0	left
1	right
2	b_led
3	left

Figure 10. Assigning trigger IDs for the 2nd RP

- You can review the hardware/software triggers in the *Trigger Mapping* tab of the PRC configuration window (Figure 11). It will show for the RP that is selected in the Virtual Socket Manager Options tab

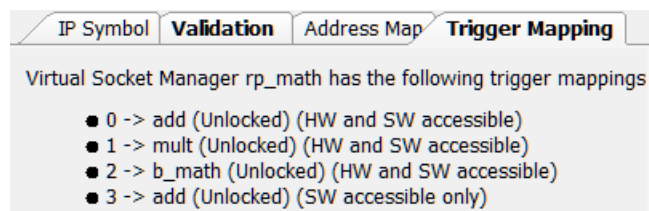


Figure 11. Triggers mapping

- You can also see the address mapping by selecting the Address Mapping tab (Figure 12)

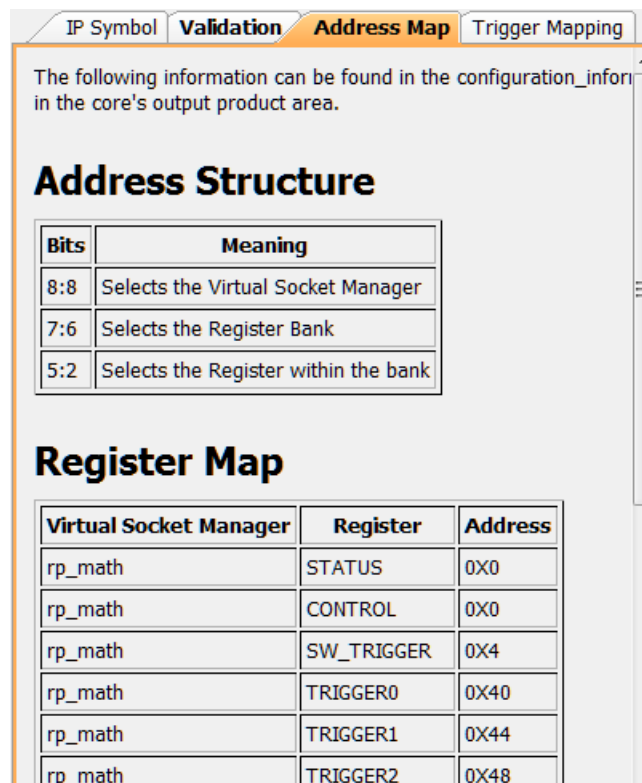


Figure 12. Address map assigned to various registers in the PRC instance

- Once the *prc* instance is configured, the connections will be made between various instances, creating some external ports (Figure 13)

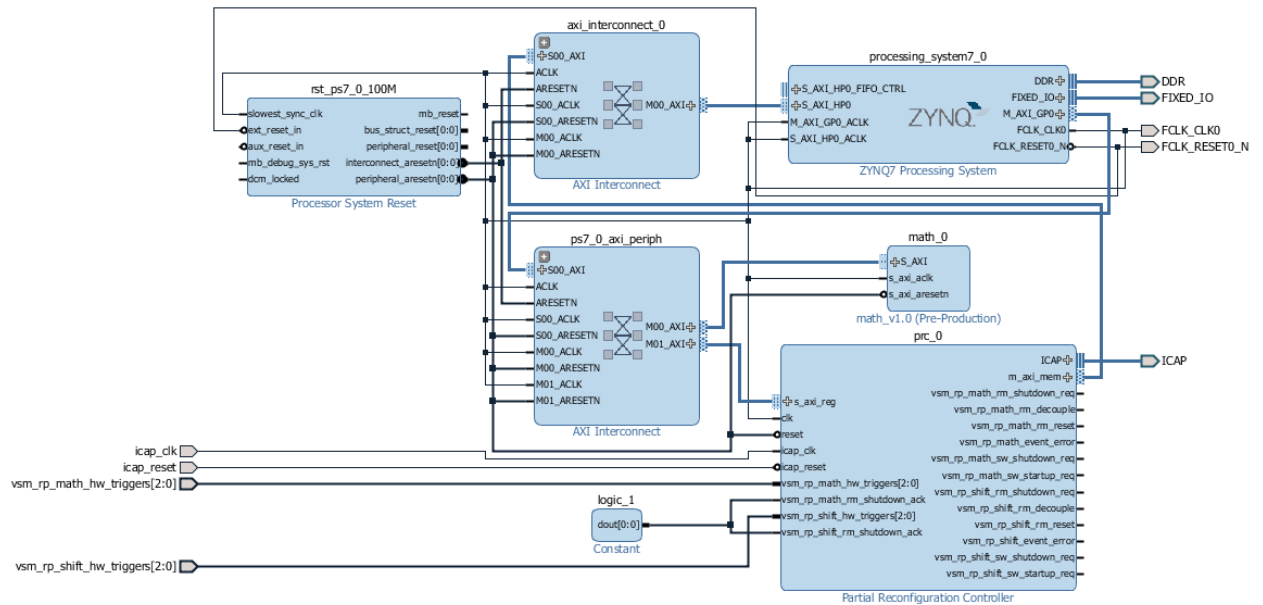


Figure 13. The complete ps7-based design (ZedBoard)

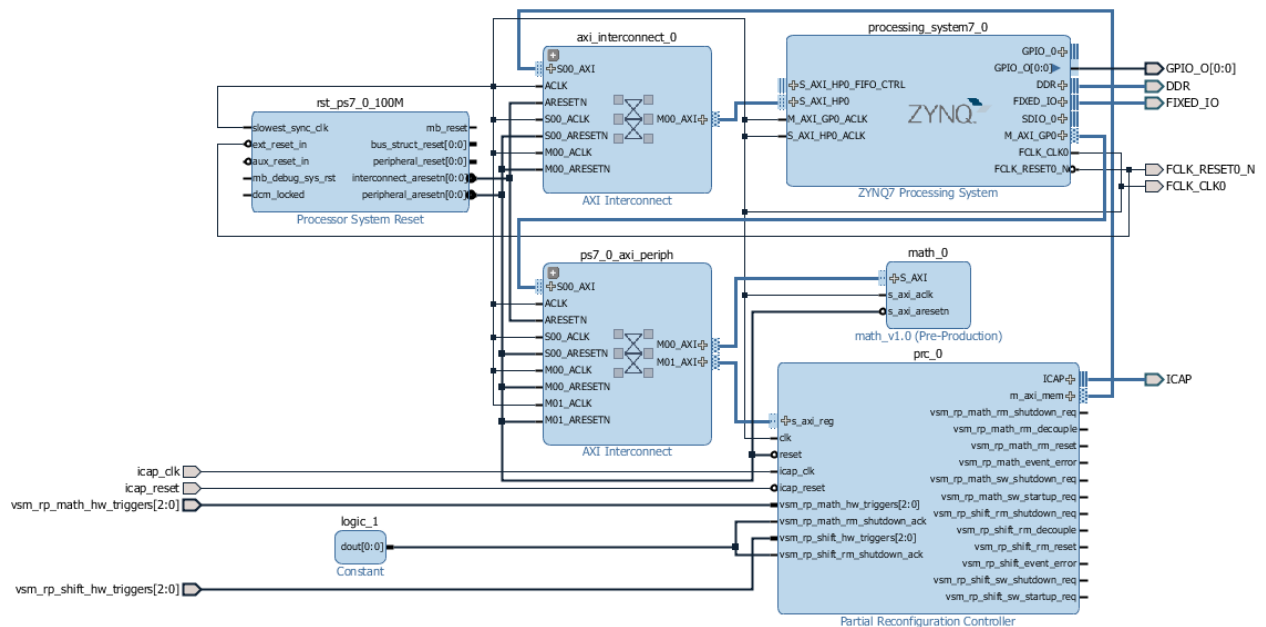


Figure 13. The complete ps7-based design (Zybo)

- 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. The block design generation will create configuration information file under the `C:\xup\PR\labs\prc_lab\prc_lab\prc_lab.srcs\sources_1\bd\system\ip\system_prc_0_0\documentation` directory. This file carries the register mapping information
- Once the wrapper file is generated, the script will add the provided *top_zed.v* (ZedBoard) or *top_zybo.v* (Zybo) and static design's rest of the modules. The design hierarchy will look like as shown in Figure 14

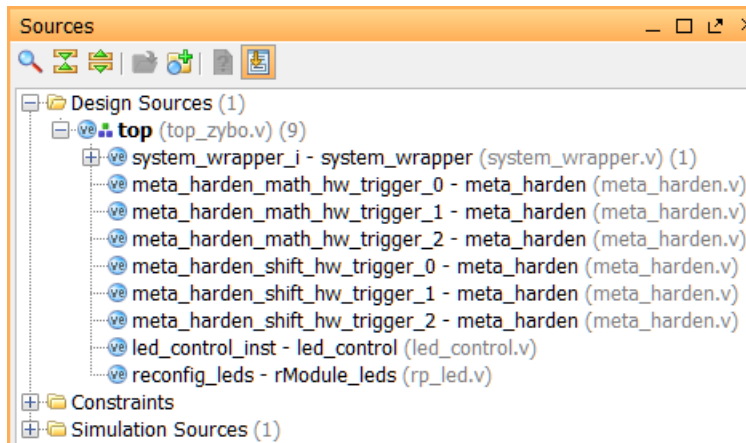


Figure 14. The design hierarchy

1-1-4. Click on the **Open Block Design** in the *Flow Navigator* and verify the above mentioned steps. Double click on the prc_0 instance to see all the customization of the PRC core.

1-1-5. Click **Run Synthesis** under the *Synthesis* group in the *Flow Navigator* to run the synthesis process. Save the project if asked.

Wait for the synthesis to complete. When done click **Cancel**.

1-1-6. Using the windows explorer, copy the **top.dcp** file from `prc_<board>_lab\prc_<board>_lab.runs\synth_1` into the *Synth\Static* directory under the current lab directory.

1-1-7. Also, copy design checkpoints for the auto_pc, math_0, prc_0, rst_ps7_0_100M, xbar, xlconstant_0 and processing_system7_0 instances. These .dcp files are also in their respective folders under the synth_1 folder of the prc_<board>_lab.runs directory. Move system_auto_pc_0.dcp, system_auto_pc_1.dcp, system_math_0_0.dcp, system_prc_0_0.dcp, system_rst_ps7_0_100M_0.dcp, system_xbar_0.dcp, system_xlconstant_0_0.dcp, and system_processing_system7_0_0.dcp to *Synth\Static* to sit alongside top.dcp

1-1-8. Close the project.

1-2. Generate the dcp for each of the RMs.

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

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

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

```
source synth_reconfig_modules_zed.tcl (for ZedBoard) OR
```

```
source synth_reconfig_modules_zybo.tcl (for Zybo)
```

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

Load Static and one RM for each RPs, and Floorplan

Step 2

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

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

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

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

```
source floorplan_design_zed.tcl (for ZedBoard) OR
```

```
source floorplan_design_zybo.tcl (for Zybo)
```

The script will do the following:

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

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

- Load following static design related dcps using the **read_checkpoint** command.

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

```
read_checkpoint -cell  
system_wrapper_i/system_i/ps7_0_axi_periph/s00_couplers/auto_pc  
Synth/Static/system_auto_pc_0.dcp
```

```
read_checkpoint -cell  
system_wrapper_i/system_i/axi_interconnect_0/s00_couplers/auto_pc  
Synth/Static/system_auto_pc_1.dcp
```

```
read_checkpoint -cell system_wrapper_i/system_i/math_0  
Synth/Static/system_math_0_0.dcp
```

```
read_checkpoint -cell system_wrapper_i/system_i/prc_0  
Synth/Static/system_prc_0_0.dcp
```

```
read_checkpoint -cell system_wrapper_i/system_i/rst_ps7_0_100M  
Synth/Static/system_rst_ps7_0_100M_0.dcp
```

```
read_checkpoint -cell  
system_wrapper_i/system_i/ps7_0_axi_periph/xbar  
Synth/Static/system_xbar_0.dcp
```

```
read_checkpoint -cell system_wrapper_i/system_i/logic_1  
Synth/Static/system_xlconstant_0_0.dcp
```

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

```
read_checkpoint -cell  
system_wrapper_i/system_i/math_0/inst/math_v1_0_S_AXI_inst/rp_ins  
tance Synth/reconfig_modules/rp_add/add_mult_synth.dcp
```

```
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
system_wrapper_i/system_i/math_0/inst/math_v1_0_S_AXI_inst/rp_instance]
```

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

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

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

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

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

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

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

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

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

Create and Implement the First Configuration

Step 3

3-1. Create and implement the first configuration.

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

```
source create_first_configuration.tcl
```

- 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_add_left/top_route_design.dcp
```

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

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

```
write_checkpoint -force -cell
system_wrapper_i/system_i/math_0/inst/math_v1_0_S_AXI_inst/rp_instance
Checkpoint/math_add_route_design.dcp
```

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

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
system_wrapper_i/system_i/math_0/inst/math_v1_0_S_AXI_inst/rp_in
stance -black_box
```

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

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

```
lock_design -level routing
```

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

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

Create and Implement the Second Configuration

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
system_wrapper_i/system_i/math_0/inst/math_v1_0_S_AXI_inst/rp_in
stance Synth/reconfig_modules/rp_mult/add_mult_synth.dcp
```

```
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_mult_right/top_route_design.dcp
```

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

```
write_checkpoint -force -cell
system_wrapper_i/system_i/math_0/inst/math_v1_0_S_AXI_inst/rp_in
stance Checkpoint/math_mult_route_design.dcp
```

```
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
system_wrapper_i/system_i/math_0/inst/math_v1_0_S_AXI_inst/rp_in
stance
```

```
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_add_left/top_route_design.dcp  
-additional {Implement/Config_mult_right/top_route_design.dcp  
Implement/Config_blank/top_route_design.dcp}
```

You should see the message indicating the `Config_add_left` configuration is compatible with `Config_mult_right`, and the `Config_add_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 full configuration and partial bitstreams by executing the following tcl script.

```
source generate_bitstreams.tcl
```

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

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

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

```
write_bitstream -file Bitstreams/Config_addleft.bit
```

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

```
write_cfgmem -format BIN -interface SMAPx32 -disablebitswap -  
loadbit "up 0
```

```

Bitstreams/Config_addleft_pblock_rp_instance_partial.bit"
Bitstreams/add.bin

close_project

```

- Generate the bitstreams for the second configuration

```

open_checkpoint Implement/Config_mult_right/top_route_design.dcp

write_bitstream -bin -file Bitstreams/Config_multiright.bit

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

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

close_project

```

- Generate the bitstreams with black boxes.

```

open_checkpoint Checkpoint/static_route_design.dcp

write_bitstream -bin -file Bitstreams/blanking.bit

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

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

close_project

```

Generate the Software Application

Step 7

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

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

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

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

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

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

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

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

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

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

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

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

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

7-3. Create an application.

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

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

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

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

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

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

- 7-3-7.** Select **TestApp_zed.c** (ZedBoard) or **TestApp_zybo.c** (Zybo) and click **Finish** to add the file to the project.

The program should compile successfully.

7-4. Create a zynq_fsbl application.

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

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

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

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

7-5. Create a Zynq boot image.

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

- 7-5-2. Click the Browse button of the Output BIF file path field, browse to *c:\xup\PR\labs\prc_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\prc_lab\prc_<board>_lab\prc_<board>_lab.sdk\zynq_fsbl\Debug* directory, select *zynq_fsbl.elf* and click **Open**.
- 7-5-4. Click **OK**.
- 7-5-1. Click again on the **Add** button of the *Boot Image partitions*, click the Browse button in the Add Partition form, browse to *c:\xup\PR\labs\prc_lab\Bitstreams* directory, select *blanking.bit* and click **Open**.
- 7-5-2. Click **OK**.
- 7-5-3. Click again on the **Add** button of the *Boot Image partitions*, click the Browse button in the Add Partition form, browse to *c:\xup\PR\labs\prc_lab\prc_<board>_lab\prc_<board>_lab.sdk\TestApp\Debug* directory, select *TestApp.elf* and click **Open**.
- 7-5-4. Click **OK**.
- 7-5-5. Make sure that the output path is *c:\xup\PR\labs\prc_lab* and the filename is *BOOT.bin*, and click **Create Image**.
- 7-5-6. Close the SDK program by selecting **File > Exit**.

Test the Design

Step 8

- 8-1. **Connect the board with one micro-USB cable (UART port). Place the board in the SD boot mode. Copy the BOOT.bin file on the SD Card. Copy the partial bin files generated in the bitstreams directory on the SD card, rename them as shown in the table below, and place the SD card in the board. Power On the board.**
- 8-1-1. Make sure that one micro-usb cable is connected between the UART port and the PC.
- 8-1-2. Make sure that the board is set to boot in SD card boot mode.
- 8-1-3. Using the Windows Explorer, copy the **BOOT.bin** from the *c:\xup\PR\prc_lab* directory on to a SD Card.
- 8-1-4. Copy the six partial bin files from the *bitstreams* directory.
- 8-1-5. Place the SD Card in the board and power ON the board.

8-2. Start a terminal emulator program such as TeraTerm or HyperTerminal. Select an appropriate COM port (you can find the correct COM number using the Control Panel). Set the COM port for 115200 baud rate communication.

8-2-1. Start a terminal emulator program such as TeraTerm or HyperTerminal.

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

8-2-3. Set the COM port for **115200** baud rate communication.

8-2-4. Press BTN7 on the board and see the initial PRC configuration activities followed by the menu.

At this point you can use the menu to generate the software triggers or press the push-buttons to generate the hardware triggers to partially reconfigure the RPs.

You can press **BTNL** (ZedBoard) or **BTN0** (Zybo) to reconfigure with the add functionality, press **BTNR** (ZedBoard) or **BTN1** (Zybo) with mult, **BTNU** (ZedBoard) or **BTN2** (Zybo) with left shift, and **BTND** (ZedBoard) or **BTN3** (Zybo) with right shift. When you press **BTNC** (ZedBoard) or **BTN4** (Zybo) it will reconfigure both RPs with blanking bitstreams.

SW1 will stop shifting and **SW0** will restart the shifting.

Also note that when the math RP is loaded with the blanking bitstream followed by entering the operands, the output may not 0. This is due to the fact that we have not turned ON the RESET_AFTER_RECONFIG option (see the used fplan.xdc file) (In earlier labs where it was turned ON, the output was 0).

8-2-5. Close Vivado by selecting **File > Exit**.

8-2-6. Power OFF the board.

Conclusion

This lab showed you how the partial reconfiguration controller (PRC) can be used to generate software triggers as well as use hardware triggers to reconfigure RPs. The software application can assign and change the triggers under the software control. The PRC has the ICAP port which can be connected to the ICAP resource. The controller also has the memory interface which can be used to load the partial bitstreams. The designation memory used in this lab is DDR but it can be non-volatile FLASH memory.