Reconfiguring Using AXI HWICAP IP Lab

Introduction

In this lab, you will use Vivado IPI and Software Development Kit to create a reconfigurable peripheral using the Zynq. You will use Vivado IPI to create a top-level design, which includes the processor system as a sub-module which instantiates a reconfigurable partition embedded in an AXI-Lite IP. In order to reconfigure the RP, an AXI HWICAP IP is used which in turn uses ICAP. In the software you will disable PCAP so you can use ICAP to reconfigure the RP. You will use either ZedBoard and/or Zybo to verify the design in hardware using a SD card to initially configure the FPGA, and then partially reconfigure the device using ICAP under user software control.

Objectives

After completing this lab, you will be able to:

- Use Tcl script to generate a Vivado IPI design, create a wrapper file from it and generate the design checkpoint
- Use Vivado's bottom-up methodology to synthesize the necessary RMs
- Floorplan the design
- Add the desired RMs
- Create multiple configurations
- Implement the design and generate full and partial bitstreams for various configurations
- Develop the software that will disable PCAP and then enable ICAP to reconfigure RP
- Download bitstreams to demonstrate a working partial reconfigurable design

Design Description

The purpose of this lab exercise is to implement a design that can be dynamically reconfigurable using PCAP resource and PS sub-system. **Figure 1** shows the system. The system consists of one peripheral (math functions), having two unique capabilities (addition and multiplication). User verifies functionality using the user application. The dynamic modules are reconfigured using the ICAP resource available through the AXI HWICAP IP.

The design is shown in Figure 1.

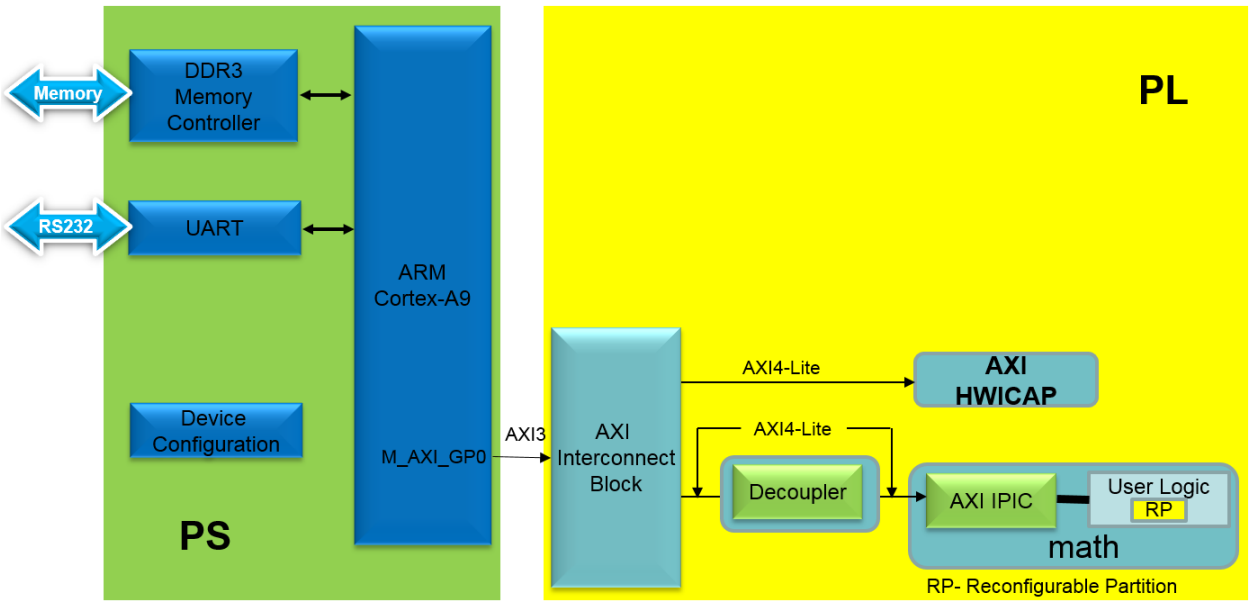
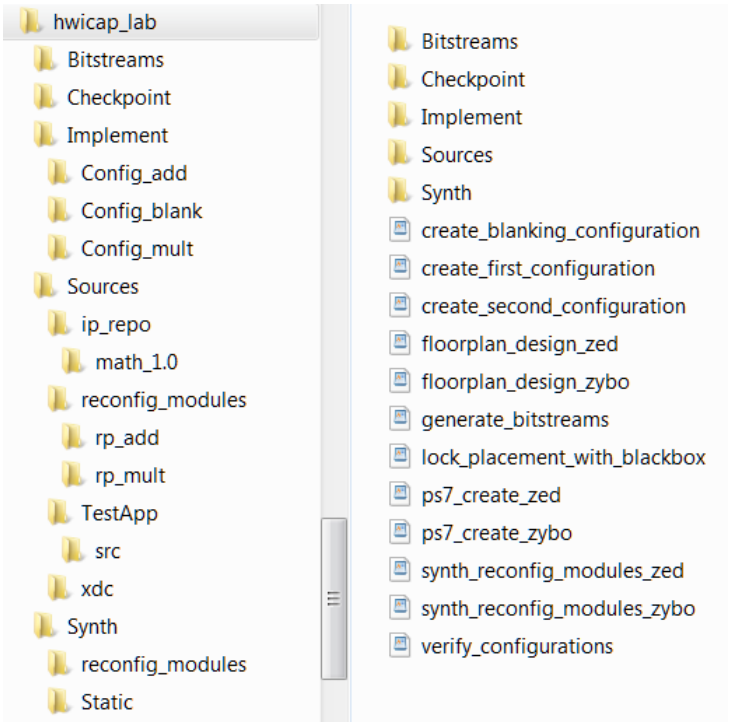


Figure 1. The design

The directory structure is as shown here:

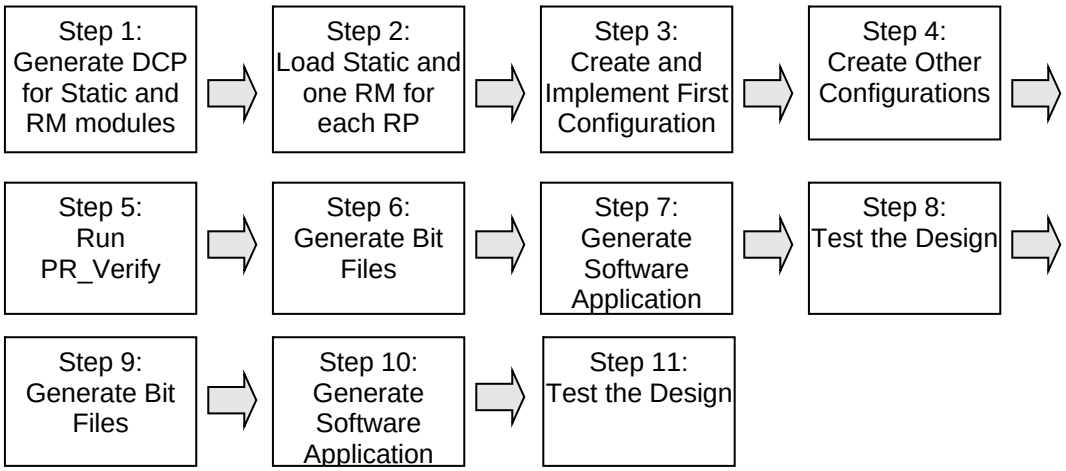


The Sources directory provides the math processor core (in ip_repo directory), source file for *add* and *mult* functions (in reconfig_modules > rp_add and reconfig_modules > rp_mult directories), software application (in TestApp directory), and a place holder for the floorplan constraints (in xdc directory). The Synth and its sub-directories structure will hold the synthesized checkpoints, the Implement and its sub-directories will hold the implemented configurations, the Checkpoint will hold the static, and the two configuration checkpoints, and the Bitstreams directory will hold the generated full and partial bitstreams. In the home directory, there are several Tcl scripts which will perform several tasks including the processor system creation and the bottom-up synthesis of the reconfigurable modules.

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 RM Modules Step 1

1-1. Start Vivado and execute the provided Tcl script to create the design check point for the static 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/hwicap_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*, instantiate ZYNQ PS with SD 0 and UART 1, 1-bit EMIO interfaces enabled. It will also enable GP0 interface along with FCLK0 and RESET0_N ports. The provided math IP will then be instantiated followed by instantiation of AXI HWICAP and PR Decoupler IP. The AXI HWICAP IP will then be configured to instantiate the STARTUP block as we don't need to wait for some other device to configure, eliminating the need to connect eos_in input. It will then create a top-level wrapper file called system_wrapper.v which instantiates the system.bd (the block design).

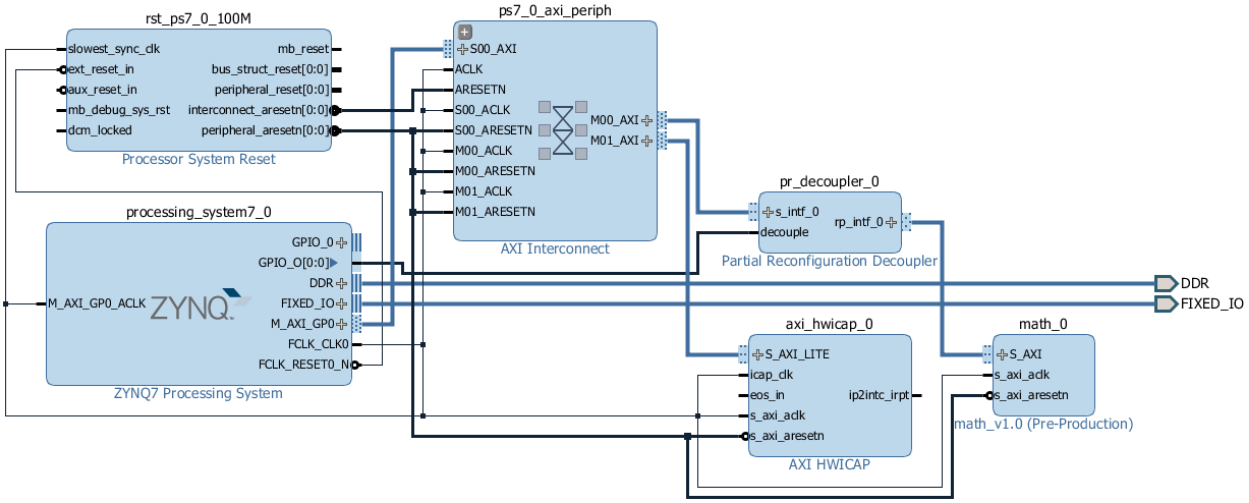


Figure 3. The system block design

1-1-4. Click **Run Synthesis** under the *Synthesis* group in the *Flow Navigator* to run the synthesis process.

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

1-1-5. Using the windows explorer, copy the **system_wrapper.dcp** file from **hwicap_<board>_lab\hwicap_<board>_lab.runs\synth_1** into the *Synth\Static* directory under the current lab directory. Use **zed** (ZedBoard) or **zybo** (Zybo) for <board>.

1-1-6. Copy design checkpoints for the auto_pc, math_0, pr_decoupler_0, rst_ps7_0_100M, axi_hwicap_0, xbar_0, and processing_system7_0 instances. These .dcp files are also in their respective folders under the synth_1 folder of the hwicap_<board>_lab.runs directory. Move system_auto_pc_0.dcp, system_math_0_0.dcp, system_pr_decoupler_0_0.dcp, system_rst_ps7_0_100M_0.dcp, system_axi_hwicap_0_0.dcp, system_xbar_0.dcp, and system_processing_system7_0_0.dcp to *Synth**Static* to sit alongside system_wrapper.dcp

1-1-7. Close the project. Save the project if asked.

1-2. Since we have RMs in HDL format, we need to synthesize them and generate the dcp for each of the RMs. The generated dcps should be stored in appropriate directories so they can be accessed correctly; particularly, the dcp files for RM must be in separate directories as their dcp file names will be same for a given RP.

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/hwicap_lab
```

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

`source synth_reconfig_modules_zed.tcl` (for ZedBoard) OR

`source synth_reconfig_modules_zybo.tcl` (for Zybo)

This script will synthesize the HDL files for a given RM in an 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. .

1-2-3. At this point the directory content will look like shown below.

Figure 4. Synth directory hierarchy and content

Load Static and one RM for the RP in Vivado Step 2

Since all required netlist files (dcp) for the design are now available, you will use Vivado to floorplan the design, define Reconfigurable Partitions, add Reconfigurable Modules, run the implementation tools, and generate the full and partial bitstreams.

2-1. In this step you will load the static and one RM designs 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/hwicap_lab
```

2-1-2. Execute the following Tcl script to load the static design checkpoint.

```
source load_design_checkpoints.tcl
```

This script will do the following:

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

  ```
  open_checkpoint Synth/Static/system_wrapper.dcp
  ```

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

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

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

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

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

  ```
  read_checkpoint -cell system_i/pr_decoupler_0
  Synth/Static/system_pr_decoupler_0_0.dcp
  ```

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

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

- Load the IP checkpoint for the auto pc by using the **read_checkpoint** command.

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

- Load the IP checkpoint for the axi hwicap by using the **read_checkpoint** command.

  ```
  read_checkpoint -cell system_i/axi_hwicap_0
  Synth/Static/system_axi_hwicap_0_0.dcp
  ```

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

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

2-1-3. Load one RM for the RP by using the **read_checkpoint** command.

```
read_checkpoint -cell
system_i/math_0/inst/math_v1_0_S_AXI_inst/rp_instance
Synth/reconfig_modules/rp_add/add_mult_synth.dcp
```

You can now see the design structure in the Netlist pane with an RM for the *rp_instance* module loaded

2-1-4. Define each of the loaded RMs (submodules) as partially reconfigurable by setting the **HD.RECONFIGURABLE** property using the following command.

```
set_property HD.RECONFIGURABLE 1 [get_cells
system_i/math_0/inst/math_v1_0_S_AXI_inst/rp_instance]
```

2-1-5. Save the assembled design state for this initial configuration (Is this required or optional) using the following command.

```
write_checkpoint Checkpoint/top_link_add.dcp
```

2-1-6. Read the previously created floorplan which defines the RP region and sets the Pblock property RESET_AFTER_RECONFIG=TRUE on the Pblock, and turns **ON** the *SNAPPING_ON* property.

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

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

Create and Implement First Configuration Step 3

3-1. Create and implement the first Configuration.

3-1-1. Execute the following command.

```
source create_first_configuration.tcl
```

The script will do the following tasks:

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

  ```
  opt_design

  place_design

  route_design
  ```

- Save the full design checkpoint.

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

- Save checkpoints for the reconfigurable module.

  ```
  write_checkpoint -force -cell
  system_i/math_0/inst/math_v1_0_S_AXI_inst/rp_instance
  Checkpoint/rp_instance_add_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
```

3-2-2. The script will do the following tasks

- Clear out the existing RMs executing the following commands.

  ```
  update_design -cell
  system_i/math_0/inst/math_v1_0_S_AXI_inst/rp_instance -black_box
  ```

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

  ```
  lock_design -level routing
  ```

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

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

Create Other Configurations Step 4

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

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

```
source create_second_configuration.tcl
```

The script will do the following tasks:

- With the locked static design open in memory, read in post-synthesis checkpoint for the second reconfigurable module.

  ```
  read_checkpoint -cell
  system_i/math_0/inst/math_v1_0_S_AXI_inst/rp_instance
  Synth/reconfig_modules/rp_mult/add_mult_synth.dcp
  ```

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

  ```
  opt_design

  place_design

  route_design
  ```

- Save the full design checkpoint.

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

- Save the checkpoint for the reconfigurable module.

  ```
  write_checkpoint -force -cell
  system_i/math_0/inst/math_v1_0_S_AXI_inst/rp_instance
  Checkpoint/rp_instance_mult_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_i/math_0/inst/math_v1_0_S_AXI_inst/rp_instance
  ```

- 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/top_route_design.dcp -
additional {Implement/Config_mult/top_route_design.dcp
Implement/Config_blank/top_route_design.dcp}
```

You should see the message indicating the `Config_add` configuration is compatible with `Config_mult`, and the `Config_add` configuration is compatible with `Config_blank`.

- Execute the following command to close the project.

```
close_project
```

Generate Bit Files — Step 6

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

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

```
source generate_bitstreams.tcl
```

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

- Read the first configuration in the memory, using the command:

```
open_checkpoint Implement/Config_add/top_route_design.dcp
```

- Generate the full and partial bitstreams for this design.

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

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

```
close_project
```

Notice the three bitstreams will be created.

Config_add.bit – This is the power-up, full design bitstream.
Config_add_pblock_rp_instance_partial.bit – This is the partial bit file for the *adder* module.
add.bin – This is the partial bit file for the *adder* module in the bin format

- Generate full and partial bitstreams for the second configuration.

```
open_checkpoint Implement/Config_mult/top_route_design.dcp
```

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

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

close_project
```

The three files will be created.

- Generate a full bitstream with black boxes, plus blanking bitstreams for the reconfigurable modules. Blanking bitstreams can be used to “erase” an existing configuration to reduce power consumption.

```
open_checkpoint Checkpoint/static_route_design.dcp

write_bitstream -file Bitstreams/blanking.bit

write_cfgmem -format BIN -interface SMAPx32 -disablebitswap
-loadbit "up 0
Bitstreams/blanking_pblock_rp_instance_partial.bit"
Bitstreams/blank.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/hwicap_lab/hwicap_<board>_lab*, select the *hwicap_<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, do not check the *Include bitstream* checkbox 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.

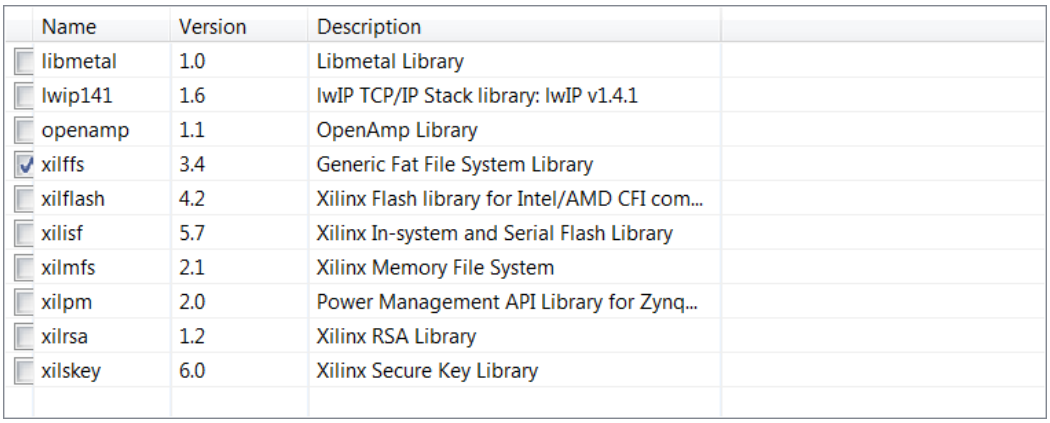
7-2. Create a Board Support Package enabling generic FAT file system library.

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.



Name	Version	Description
libmetal	1.0	Libmetal Library
lwip141	1.6	lwIP TCP/IP Stack library: lwIP v1.4.1
openamp	1.1	OpenAmp Library
xilffs	3.4	Generic Fat File System Library
xilflash	4.2	Xilinx Flash library for Intel/AMD CFI com...
xilisf	5.7	Xilinx In-system and Serial Flash Library
xilmfs	2.1	Xilinx Memory File System
xilpm	2.0	Power Management API Library for Zynq...
xilrsa	1.2	Xilinx RSA Library
xilskey	6.0	Xilinx Secure Key Library

Figure 10. Selecting the xilffs library support

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\hwicap_lab\Sources\TestApp\src* and click **OK.**

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

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

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

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

The program should compile successfully.

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

7-4. Create a zynq_fsbl application.

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

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

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

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

7-5. Create a Zynq boot image.

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

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

7-5-3. Click on the **Add** button of the *Boot image partitions,* click the Browse button in the Add Partition form, browse to **c:\xup\PR\labs\hwicap_lab\hwicap_ <board>_lab\hwicap_< 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\hwicap_lab\Bitstreams** directory, select *Config_add.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\hwicap_lab\hwicap_< board>_lab\hwicap_< 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\hwicap_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 micro-USB cable connected to the UART. Place the board in the SD boot mode. Copy the generated BOOT.bin and the partial bit files on the SD card and place the SD card in the board. Power On the board.

8-1-1. Make sure that a micro-usb cable is connected to the UART port.

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/hwicap_lab/* directory on to a SD Card.

8-1-4. Using the Windows Explorer, copy the three partial bin files in the *Bitstream* directory to *blank.bin, mult.bin,* and *add.bin* on to the SD Card

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 to display a menu.

8-2-5. Follow the menu and test various reconfigurations.

Typing 1, 2, or 3 at the menu will let you partially reconfigure the multiplication, addition, blanking functionality respectively. Typing 4 will let you enter the operands and provide you the result.

Try various reconfigurations and enter operands after each reconfiguration to verify that the design indeed works.

When blanking bitstream is loaded, the result is 0.

8-2-6. Close Vivado by selecting **File > Exit**.

8-2-7. Power OFF the board.

Conclusion

This lab showed you steps involved in creating a processor system having a reconfigurable module placed down deeper in a hierarchy in an AXI-Lite IP using Vivado IPI. The design used the AXI HWICAP IP since we wanted to reconfigure the RP using the ICAP. Full bitstream as well as partial reconfiguration bitstreams were generated by going through the PR flow. You also learned how to generate the boot image as well as how to convert the partial bit files to bin format. You verified the functionality either using ZedBoard and/or Zybo.