ECEN 449: Microprocessor System Design
Department of Electrical and Computer Engineering
Texas A&M University

Prof. Sunil P Khatri
(Lab exercise created and tested by Ramu Endluri, He Zhou and Sunil P Khatri)

Laboratory Exercise #3

Creating a Custom Hardware IP and Interfacing it with Software

Objective

The purpose of lab this week is to familiarize you with the process of creating and importing a custom IP module for a Zynq Processing System based system. We will be using the 'Create and Package IP' in Vivado to develop a custom peripheral for performing integer multiplication. We will then integrate the integer multiplication peripheral into a microprocessor system and develop software to interact with the peripheral using the SDK. This lab serves as a simple hardware/software co-design example.

System Overview

The microprocessor system you will build in this lab is depicted in Figure 1. In the previous lab a soft processor Microblaze was used, however in this lab you will use the ARM Cortex A9 processor in the Zynq Chip on ZYBO board. The Zynq chip is divided into Processing System(PS) and Programming Logic(PL). PS has a dual-core ARM Cortex-A9 processor and PL uses Xilinx 7 series FPGA logic cells. In place of the GPIO modules, utilized in the last lab, is a multiplication block, which represents the integer multiplication peripheral you will create. The UART in Figure 1 will be used to connect the USB-UART port(J11) on the ZYBO board to the workstation computer via a cable which will display the output of the software executing on the PS. The software you will develop in this lab will provide proof of operation for your multiplication peripheral.

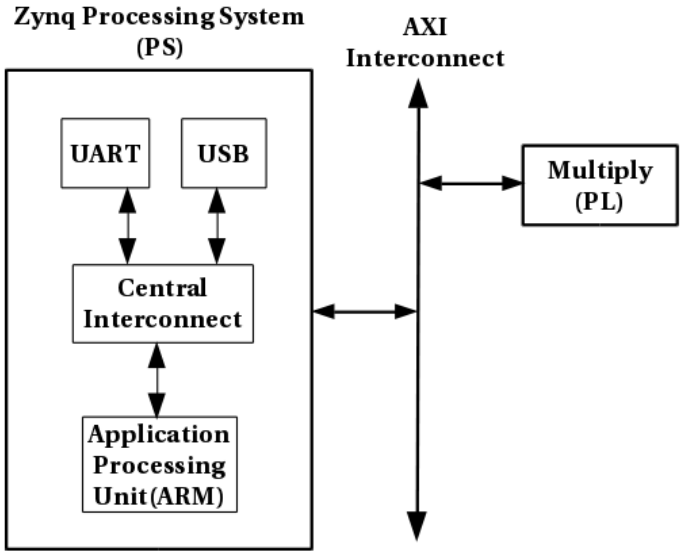

Figure 1: Zynq System Diagram

Procedure

Create Zynq Base System

1. To begin, open vivado and create a new project as shown in previous lab with one exception. In the Default part window, click on 'Boards' and select 'Zybo' as shown in Figure 2. As discussed in Lab 1 we will select hardware using the 'Board' tab from now on.

2. Click on 'Create Block Design' and name the design 'multiply'. In the diagram, add 'ZYNQ7 Processing System' IP as in Figure 3.

3. Do not select 'Run Block Automation' as in previous lab. Double click on the PS IP to open 'Re-customize IP' window. Download the 'ZYBO_zynq_def.xml' file from /home/faculty/shared/ECEN449 and save this XML file somewhere under your Linux account by using the 'cp' command. In the 'Re-customize IP' window, click on 'Import XPS Settings', and import the XML file you just downloaded. Then click on 'Peripheral I/O Pins' tap, and uncheck all the peripheral I/O pins.

4. In the 'Re-customize IP' window, click on 'Peripheral I/O Pins'. To build the hardware depicted in Figure3. Enable 'UART 1' by clicking on the check box. Double check and make sure that only the

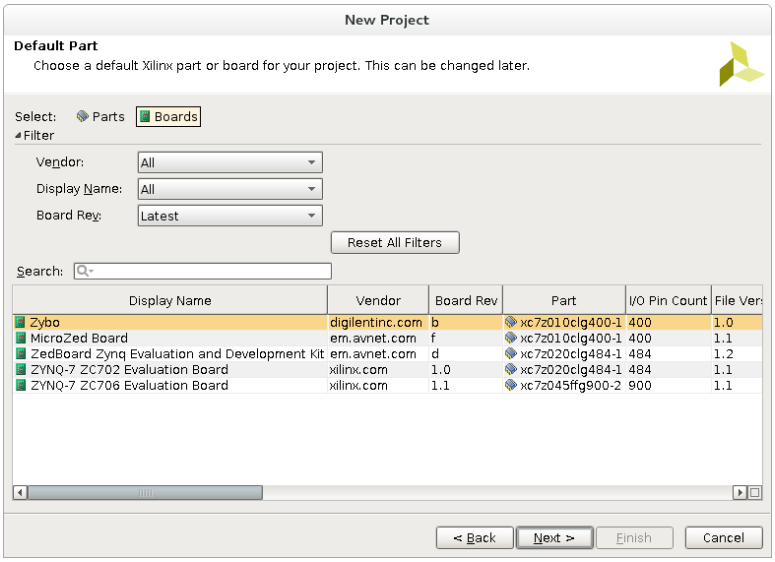

Figure 2: ZYBO Board

'UART 1' is selected.

5. Click on 'Zynq Block Design' tab and observe the diagram. Note that the 'Application processing Unit' which represent the ARM processors on the ZYBO board is connected to 'UART 1' through a 'Central Interconnect' block. Click 'OK'

6. PS is ready and the next part is to create the 'multiply' IP. Click on 'Tools' and select 'Create and Package IP'. This will open 'Create and Package New IP' window and click on 'Next'. Now select 'Create a new AXI4 peripheral' and click on 'Next'.

7. A window will appear prompting you to assign a name and version number to your peripheral(Figure4). Name the peripheral 'multiply' and leave the version number as default (i.e 1.0). You can enter a short description of your peripheral if you would like in the 'Description' field.

8. The next window will ask us to select 'Interface'. Leave the default values
   Interface type : Lite
   Interface mode: slave
   Data Width: 32
   Number of Registers: 4
   We will need only three registers for the 'multiply' IP however the minimum number of registers allowed is 4 (see Figure 5, Click on 'Next'. You will see a summary of the IP we have created.

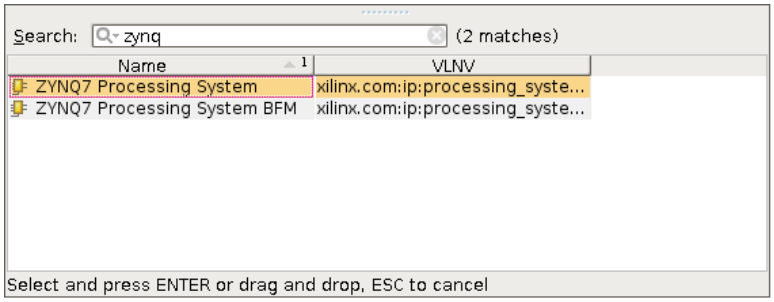

Figure 3: Zynq Processing System

We still need to add the functionality of the multiplier to this IP. Check 'Edit IP' and click 'Finish'. Another Vivado window will open which will allow you to modify the peripheral that we created.

9. At this point, the peripheral that has been generated by Vivado is an AXI lite slave that contains 4 x 32 bit read/write registers. We want to add our multiplier code to the IP and modify it so that two of the registers connect to the multiplier inputs and another register connects to the multiplier output.

Edit 'mutiply' IP and Import it to PS

1. Open the new Vivado window that contains the peripheral we created (not the base project). Review the information under the various tabs in the Package IP tab.

2. In the sources window, expand 'multiply_v1_0' and double click on the 'multiply_v1_0_S00_AXI.v' file to open it.

3. Examine the verilog code in file and try to understand each function, especially how the 'read' and 'write' functions are implemented. Comment out any code that writes to 'slv_reg2'(i.e.' slv_reg2 <='). This will deactivate the write capabilities to the third software register which is our multiplier output. Remember that we cannot have two drivers for a particular register unless we add multiplexing logic.

4. While the 'multiply_v1_0_S00_AXI.v' file is still open, locate the '// Add user logic here' line and insert the following code:

```
reg [0 : C_S_AXI_DATA_WIDTH-1] tmp_reg ;
always @( posedge S_AXI_ACLK ) begin
  if( S_AXI_ARESETN == 1'b0 ) begin
          slv_reg2 <=0;
          tmp_reg <=0;
  end
  else begin
```

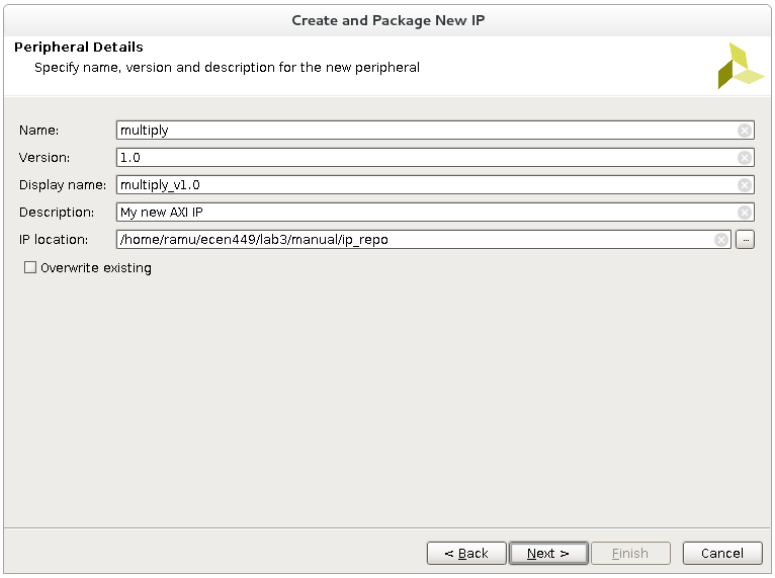

Figure 4: Create New IP

```
            tmp_reg <= slv_reg0 * slv_reg1;
            slv_reg2 <= tmp_reg ;
    end
end
```

Examine the above code, comment it appropriately, and save it.

5. In Package IP tab, click on 'Review and Package' and select 'Re-package'. Now Vivado will repackage the IP with added functionality. When Vivado finishes packaging the IP, close the project.

6. At this point in the lab, we have used Vivado to generate template hardware peripheral files for our multiplication peripheral based on our specifications. We have also added user logic in Verilog to our template for the multiplication functionality of our peripheral. We are now ready to import our peripheral into Vivado and add it to our PS system.

7. Now, select the Vivado window which has the PS system. Open the diagram tab and add 'multiply' IP to the PS system. Select 'Run Connection Automation' and in the prompted window select 'All Automation' and click 'OK'. Once automation is completed, a layout is generated. Right click on the diagram and select 'Regenerate Layout' to re draw the layout design. The layout is shown in Figure6. A 'Processor System Reset' block is also created which synchronizes the peripheral and interconnect reset to clock.

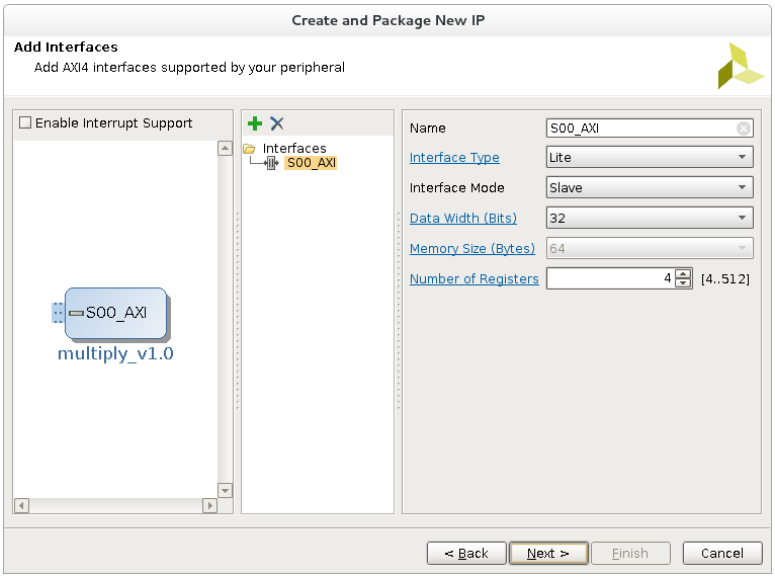

Figure 5: IP Interface Options

8. In the source tab, right click on your block design and select ‘Create HDL wrapper’. In the ‘Create HDL wrapper’ window select ‘Let Vivado manage wrapper and auto update’. This will create the top module for the blocks in the block diagram. Click ‘OK’

9. Now we have the PS system and multiply IP ready. It is time to generate the bitstream. In the Flow Navigator, select ‘Generate Bit Stream’. Ignore any critical messages during the process.

10. Once the bit generation is complete, it is time to write an application and test the multiply IP.

11. Export the design including bit stream as shown in previous lab.

Launch SDK and write multiplication test application.

1. Currently, our hardware should be ready to go. Next step is to create an application to test our ‘multiply’ IP peripheral. Open the SDK and click on ‘File’ and select ‘New’ -> ‘Application Project’. In the new project window, give a name(eg. multiply_test) to the project and leave the default values for the remaining fields. Click ‘Next’ and select ‘Hello World!’ and ‘Finish’. This step will generate the necessary templates files for our PS on the ZYBO board.

2. In the project explorer, expand ’multiply_test’. Under the src folder, open ‘helloworld.c’ file and examine the code generated.

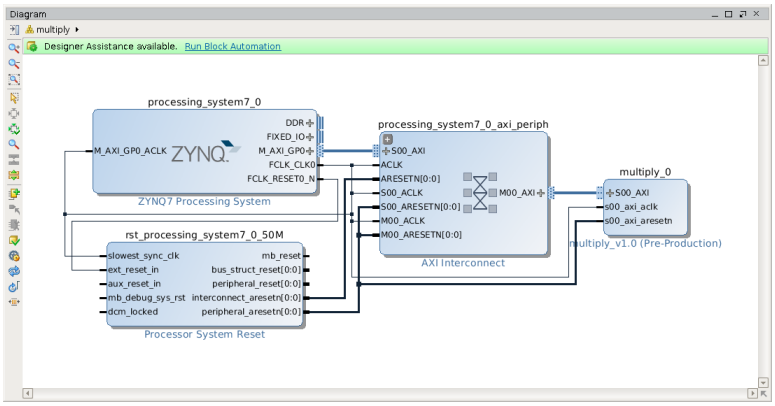

Figure 6: Layout of the Design

3. Edit the 'helloworld.c' source file to write values to the registers 'slv_reg0' and 'slv_reg1' and read the multiplication result from 'slv_reg2'. The value stored in each of these three registers should be printed to the terminal using printf.

4. Once you have written the source file, save it under src folder. Click on 'Xilinx Tools' and select 'Program FPGA' to program the bitstream file to the ZYBO board. Click 'Program'. Under 'multiply_test' expand 'binaries'. Right click on 'multiply_test.elf', select 'Run As' and click on 'Launch on Hardware(GDB)'. This command creates a configuration file which we can use to run our application. Vivado will launch the application on the ZYBO board.

5. To see the output of the printf statements in our code, we must use 'picocom', a serial console application on the CentOS machines. Open a terminal window and type the following:

   $ source /softwares/Linux/xilinx/Vivado/2015.2/settings64.sh
   $ picocom -b 115200 /dev/ttyUSB1

   If everything is correct, you will see text being printed to the serial console. Demonstrate your progress to the TA.
   The following Hints will help:

   - You will need to include the xparameters.h and multiply.h in the source file. SDK will dislay the files included in your source code in the outline window to the right side.
   - Look for functions to read and write to a register in multiply.h
   - The *RegOffset* value for 'slv_reg0' is 0, and the four 32-bit registers are consecutively located in memory.

- Use a for-loop to write different values (varying from 0 to 16) in 'slv_reg0' and 'slv_reg1' and read the corresponding multiplication result from 'slv_reg2'.
- To read or write to slave registers, make sure that the base address is of the type 'Unsigned 32-bit integer (u32)'

Deliverables

1. [4 points.] Demo the working multiplier to the TA.

   Submit a lab report with the following items:

2. [5 points.] Correct format including an Introduction, Procedure, Results, and Conclusion.
3. [4 points.] Commented C file.
4. [2 points.] Commented Verilog code.
5. [2 points.] The output of the terminal (picocom).
6. [3 points.] Answers to the following questions:
   (a) What is the purpose of the tmp_reg from the Verilog code provided in lab, and what happens if this register is removed from the code?
   (b) What values of 'slv_reg0' and 'slv_reg1' would produce incorrect results from the multiplication block? What is the name commonly assigned to this type of computation error, and how would you correct this? Provide a Verilog example and explain what you would change during the creation of the corrected peripheral.