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:
Kushagra Gupta, Cheng-Yen Lee, Abbas Fairouz, Ramu Endluri, He Zhou,
Andrew Douglass and Sunil P. Khatri

Laboratory Exercise #3

Creating a Custom Hardware IP and Interfacing it with Software

Jan2024

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 Z7-10 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(J12) on the ZYBO Z7-10 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.

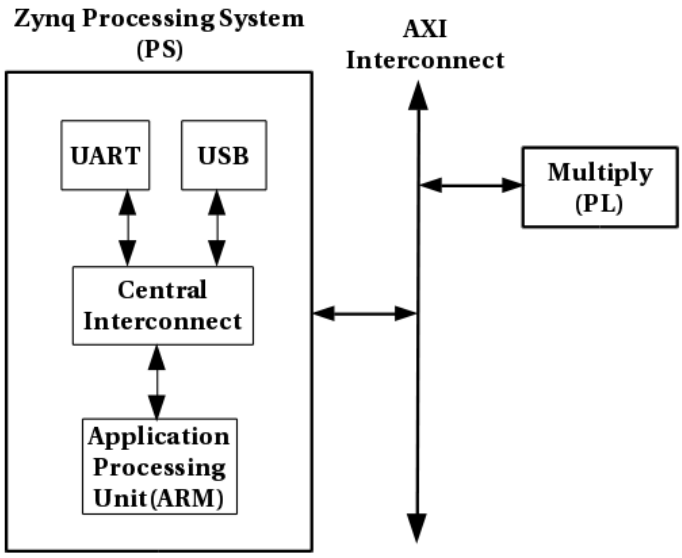

Figure 1: Zynq System Diagram

Procedure

1. Create Zynq Base System

   (a) 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 Z7-10' as shown in Figure 2. As discussed in Lab 1 we will select hardware using the 'Board' tab from now on. If you don't find the board, click on 'Refresh'.

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

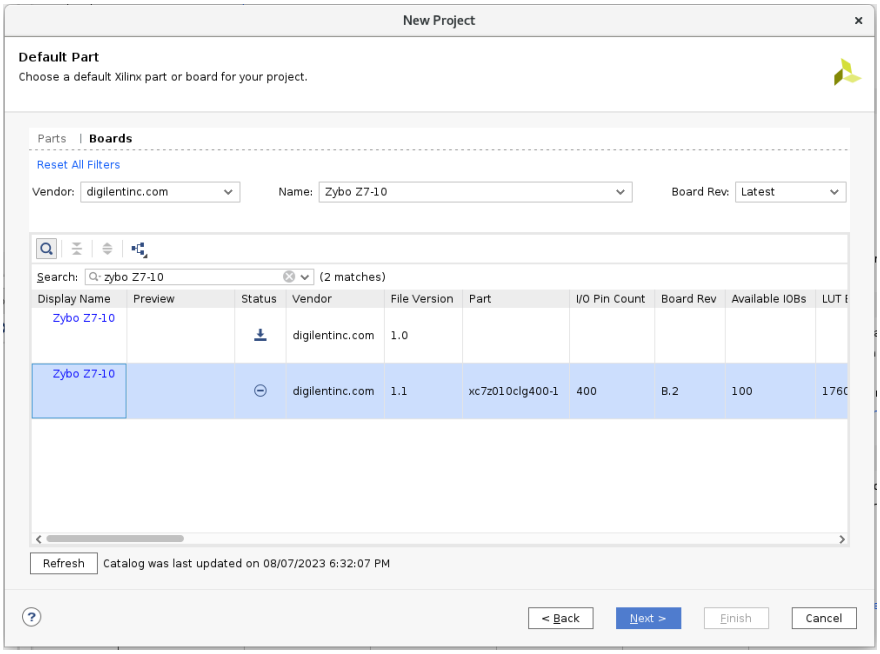

Figure 2: ZYBO Z7-10 Board

(c) 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_Z7_B2_.tcl' file from /mnt/lab_files/ECEN449 and save this TCL file somewhere under your Linux account by using the 'cp' command. In the 'Re-customize IP' window, click on 'Presets → Apply Configuration', and import the TCL file you just downloaded. Then click on 'Peripheral I/O Pins' tap, and uncheck all the peripheral I/O pins.

(d) 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 'UART 1' is selected.

(e) Click on 'Zynq Block Design' tab and observe the diagram. Note that the 'Application processing Unit' which represent the ARM processors on the ZYBO Z7-10 board is connected to 'UART 1' through a 'Central Interconnect' block. Click 'OK'

(f) PS is ready and the next part is to create the 'multiply' IP. Click on 'Tools' and select 'Create and Package New 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'.

(g) A window will appear prompting you to assign a name and version number to your peripheral (Figure 4). Name the peripheral 'multiply' and leave the version number as default (i.e 1.0).

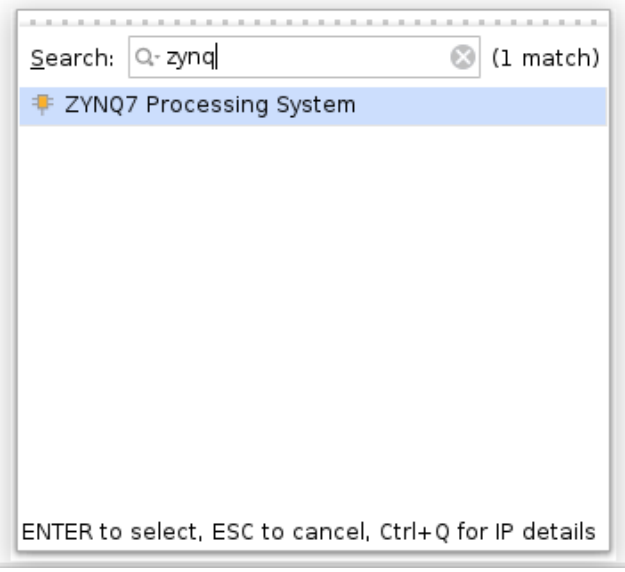

Figure 3: Zynq Processing System

You can enter a short description of your peripheral if you would like in the 'Description' field. Then, click on 'Next'.

(h) 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. 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.

(i) 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.

2. Edit 'multiply' IP and Import it to PS

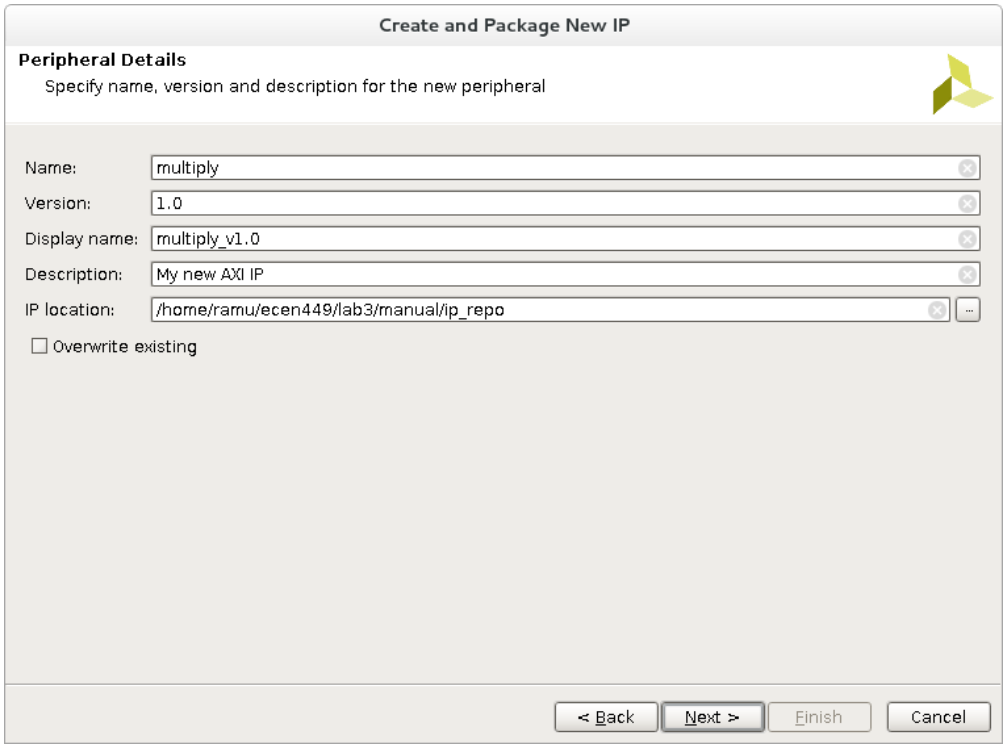

Figure 4: Create New IP

(a) 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.

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

(c) 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.

(d) 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;
```

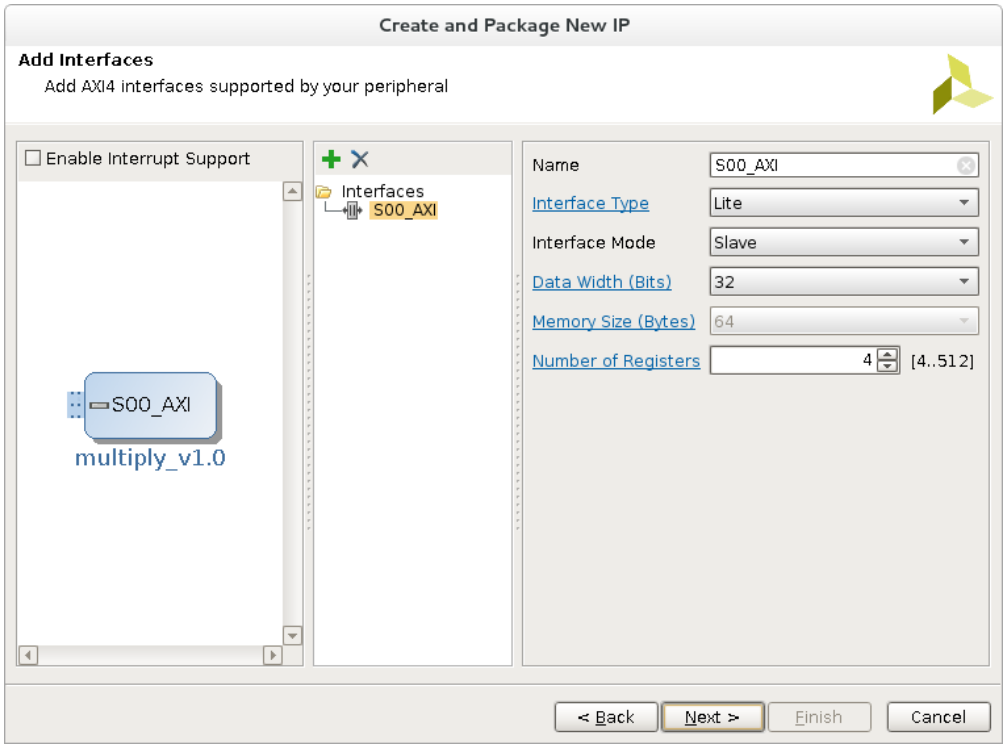

Figure 5: IP Interface Options

```
            tmp_reg <=0;
        end
        else begin
            tmp_reg <= slv_reg0 * slv_reg1;
            slv_reg2 <= tmp_reg ;
        end
    end
```

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

(e) 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.

3. 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.

4. 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 Figure 6. A 'Processor System Reset' block is also created which synchronizes the peripheral and interconnect reset to clock.

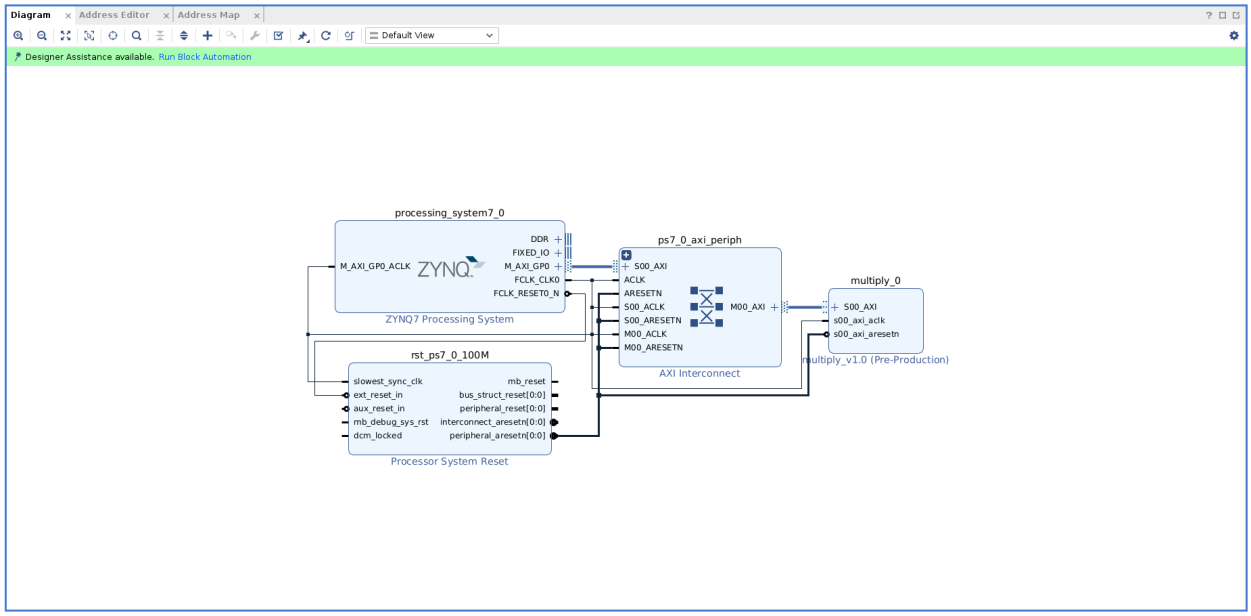

Figure 6: Layout of the Design

5. 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'

6. Now we have the PS system and multiply IP ready. It is time to generate the bitstream. In the Flow Navigator, select 'Generate Bitstream'. Now, save 'multiply' project, and ignore any critical messages during the process.

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

8. Export the design including bitstream as shown in the previous lab.

9. Launch Vitis (Tools $\rightarrow$ Launch Vitis IDE) and write multiplication test application.

(a) Currently, our hardware should be ready to go. The next step is to create an application to test our 'multiply' IP peripheral. Launch Vitis and set the workspace as lab3 project directory. Click on 'File' and select 'New' → 'Application Project'. Click 'Next' on the welcome page.

(b) In the new application project window, select the tab 'Create a new platform from hardware (XSA)' and click on 'Browse' to select the exported hardware (.xsa file). Click 'Next'.

(c) In the next window, give a name (eg. multiply_test) to the application project, and click 'Next'. Leave the default values for the domain in the next window and click 'Next'. Select 'Hello World' and click on 'Finish'. This step will generate the necessary templates files for our PS on the ZYBO Z7-10 board.

(d) In the explorer view, expand 'multiply_test_system' → 'multiply_test'. Under the 'src' folder, open 'helloworld.c' file and examine the code generated.

(e) 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.

(f) Once you have written the source file, save it under src folder. Build the hardware platform by right clicking on 'multiply_wrapper' and selecting 'Build project'. Similarly build the software system ('multiply_test_system'). Once the build succeeds, you will see the application binary file (multiply_test.elf) under 'multiply_test_system' → 'multiply_test' → 'Binaries' in the explorer view.

(g) Click on 'Xilinx' → 'Program Device' to open the program device window. Click on 'Program' to program the bitstream file to the ZYBO Z7-10 board. Right click on 'multiply_test' and select 'Run As' → 'Launch Hardware (Single Application Debug)' to run our application on the ZYBO Z7-10 board.

(h) 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 /opt/coe/Xilinx/Vivado/2022.1/settings64.sh
$ picocom -b 115200 /dev/ttyUSB1
```

(To exit from picocom, press Ctrl-a then Ctrl-x).

Note: you need to make sure that the baud rate of the Xilinx SDK is 115200.

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. Vitis will display 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)'
- Do not remove the functions that initialize and clean the board in your C program. They are essential for your code to operate properly.
- If you close the multiply IP block, you can open it again by right clicking on the multiply IP block and selecting 'Edit in IP Packager'.

Deliverables

1. [7 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. [2 points.] The output of the terminal (picocom).
4. [6 points.] Answers to the following questions:
   (a) Recall that 'slv_reg0', 'slv_reg1', and 'slv_reg2' are all 32-bit registers. 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.
   (b) In this exercise, we wrote the multiplier and the multiplicand to the input registers, followed by a read from the output register for the multiplication result. Is it possible that we end up reading the output register before the correct result is available? Why?
   (c) While creating the multiply IP, we kept the interface mode as "Slave". Suppose we change this mode to "Master", do you think it will impact our experiment? Why?