Mauro Lopez

LAB 3 Report

ECEN-449

Section 502

September 27, 2018

Introduction:

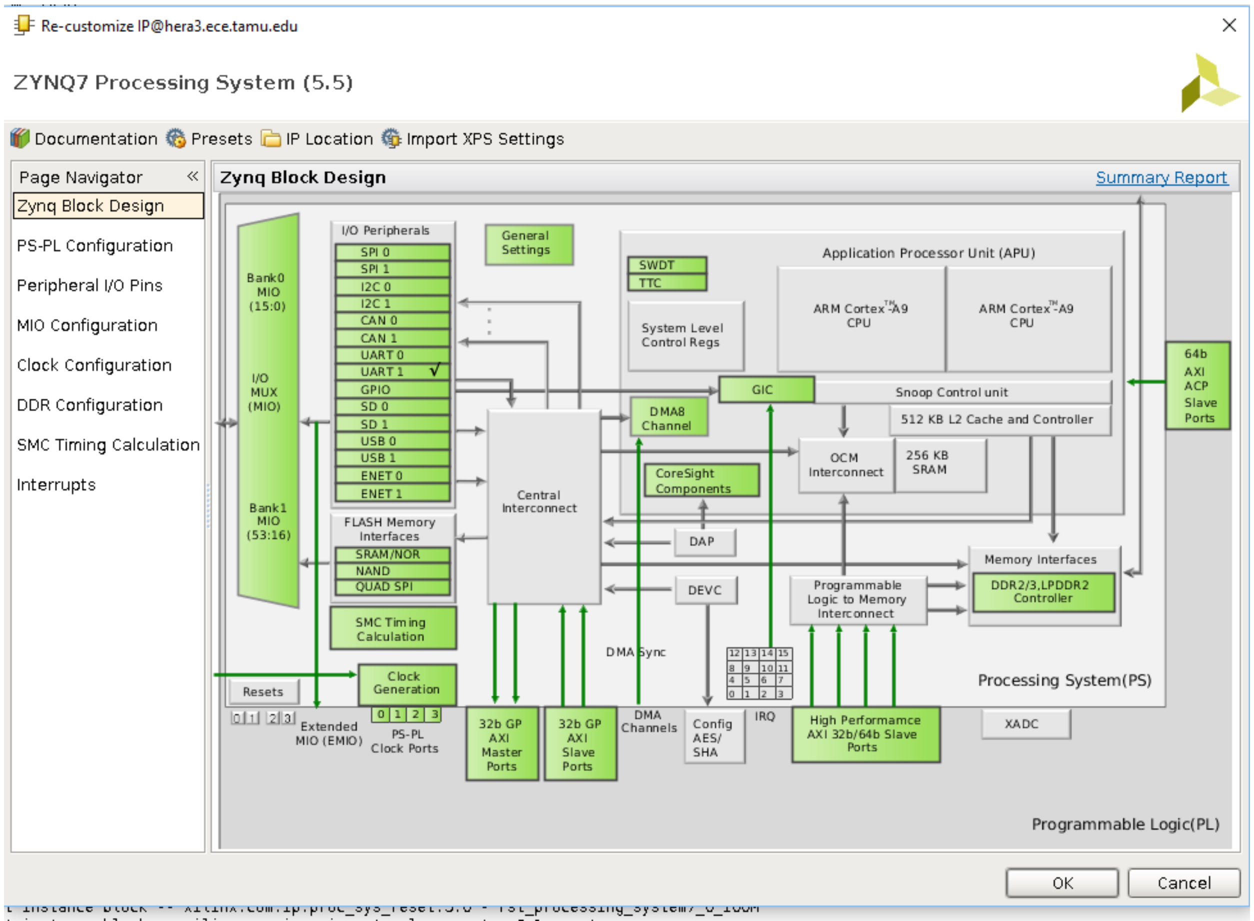
In this lab we are going to approach a new method of implementing software into hardware. We will be doing this by implementing a multiply program with ARM and C. To perform these tasks, we will be using vivado and SDK.

Procedure:

like our previous labs we will be creating a new project in vivado. The only difference is that we will be using a board, and a zybo.

Once we have our project build we can begin to design our new IP block. First, we select “create block design “, we name it “multiply”. Then we add a “ZYNQ7 Processing system” to our diagram. We will be adding our custom XPS settings by copying the file from “/home/faculty/shared/ECEN449/ecen449NeededFiles/ “. For now we will be un-selecting all of the boxes.

Once we have done everything we should end up with something similar to this:

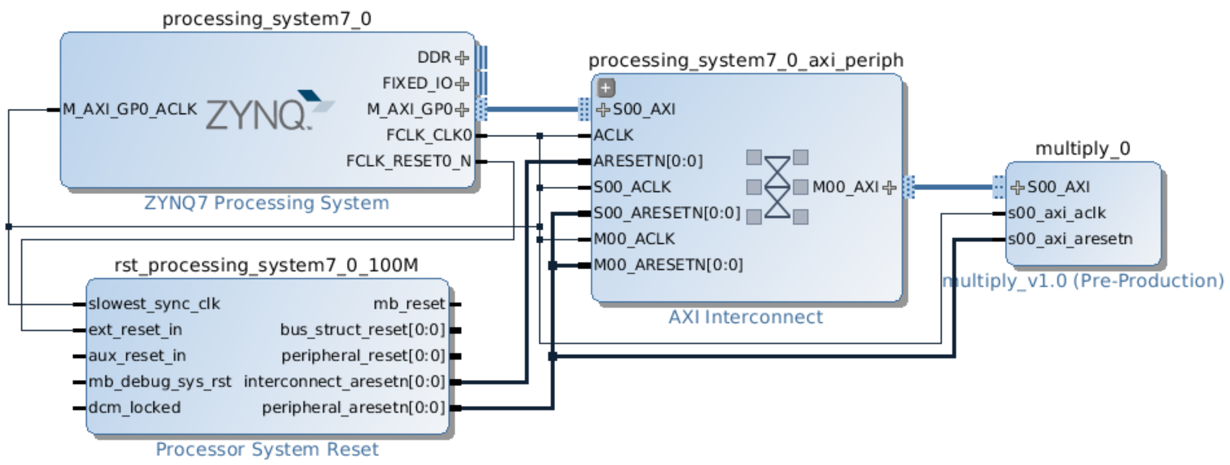


Once we have our zynq block ready, we can begin to edit our multiply IP. We will be creating a new A14 peripheral. This will create a new window in vivado, which will allow us to edit the logic that will be used. For this lab we will be using our register 2,1, and 3. This means we will have to go back and erase anything that assigns register 2.

We then can move on and add our own logic. We add the following lines to the bottom of our 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 ;

                Temp<= 0;

End

Else begin

Temp_reg <= slv_reg0 * slv_reg1;

Slv_reg2<=temp_reg;

End

End
```

Once we have type this into our code, we can re-package it. finally we go to our block diagram and select, run connection automations. Our final block diagram should look something like this:



We then create an HDL wrapper. Then we generate our bit stream, and export that before launching SDK.

Once we are in SDK, we create a new project with a template of hello world. We can compile it, but we wont necessary get an output to our terminal. To enable our terminal, we launch a separate terminal with the following commands.

“source/softwares/Linux/xilinx/Vivado/2015.2/settings64.sh

Picocom -b 115200/deb/ttyUSB1

Once we have our terminal and program running we should have something similar to this.

```c
#include <stdio.h>
#include "platform.h"
#include "xparameters.h"
#include "multiply.h"

void print(char *str);
#define BASE_ADDRESS XPAR_MULTIPLY_0_S00_AXI_BASEADDR
//define our variable from our microprocessor

int main()
{
    init_platform();
    //unsigned long MULTIPY_ID;
    //create and define our register values.
    unsigned long slv_reg0=0;
    unsigned long slv_reg1=0;
    unsigned long slv_reg2=0;

    unsigned long X0,X1;
    unsigned long i;
    X0=3;
for(i =0; i<=16; i=i+1){ // testing loop for multiple inputs
       X1=i;
```

```
        MULTIPLY_mWriteReg(BASE_ADDRESS, 0,X0);
        slv_reg0=MULTIPLY_mReadReg(BASE_ADDRESS, 0);

        MULTIPLY_mWriteReg(BASE_ADDRESS, 4,X1);
        slv_reg1=MULTIPLY_mReadReg(BASE_ADDRESS, 4);

        slv_reg2=MULTIPLY_mReadReg(BASE_ADDRESS, 8);

         printf("slv_reg0 =  %d \n",slv_reg0);
         printf("slv_reg1 =  %d \n",slv_reg1);
         printf("slv_reg2 =  %d \n\n",slv_reg2);
         //printf("%d",i);
}

    cleanup_platform();
    return 0;
}
```

Results:

Looking at the output terminal we can see that the program was successful, by comparing their values with a computational value. For example 2*5 =10, when compare to the program it matches. We can see that the result always gets stored into the second register, so to get result, we need to have a variable assign to register 2.

Conclusion:

In this lab we learn more about software design for different software, specifically ARM. We started of by creating our block diagram and changing the logic behind our multiply IP block. We then use SDK to launch our program and test our results. We used the multiply function from Verilog to apply it to our microprocessor, instead of us making it at a gate level. Overall the lab was a success and insightful into hardware design.

Questions:

1)

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?

Temp_reg allows us to temporary store the product of Register_0 and Register_1, this will act as was way to keep our program stable. Removing this might not affect our simulations given that is running on ideal values, but on hardware it might vary, and to avoid this we add an extra latch to prevent failures.

2) **What values of ‘slv reg0’and‘slv reg1’wouldproduceincorrectresultsfromthemultiplication 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.**

We would get incorrect values for our computations when we use numbers bigger than 32 bits and multiply them, this will lead to an over flow and will return the wrong answer simply because 32 bits isn’t enough to represent that value. An example include

2,147,483,647* 2,147,483,647= x