Lab #3:

Creating Custom Hardware IP and Interfacing it with Software

Khanh Nguyen

UIN# 525000335

ECEN 449– 503

Date: February 9, 2017

INTRODUCTION:

The purpose of this lab is to help us familiar with the process of creating and importing a custom IP module for a Zynq Processing System. We developed a custom peripheral for performing integer multiplication and implement it into a micro processor then develop a software to interact with it using SDK.

PROCEDURE:

1/ Create a new project and choose Zybo as the board.

2/ Import Z ZYBO_zynq_def.xml into XPS settings in the Customize IP window.

3/ Enable UART1 and disable all other peripheral pins

4/ Create a new AXI peripheral block called ‘multiply’.

5/ Create HDL wrapper and generate bitstream then export and switch to SDK.

6/ Create a new application project Hello world and modify it.

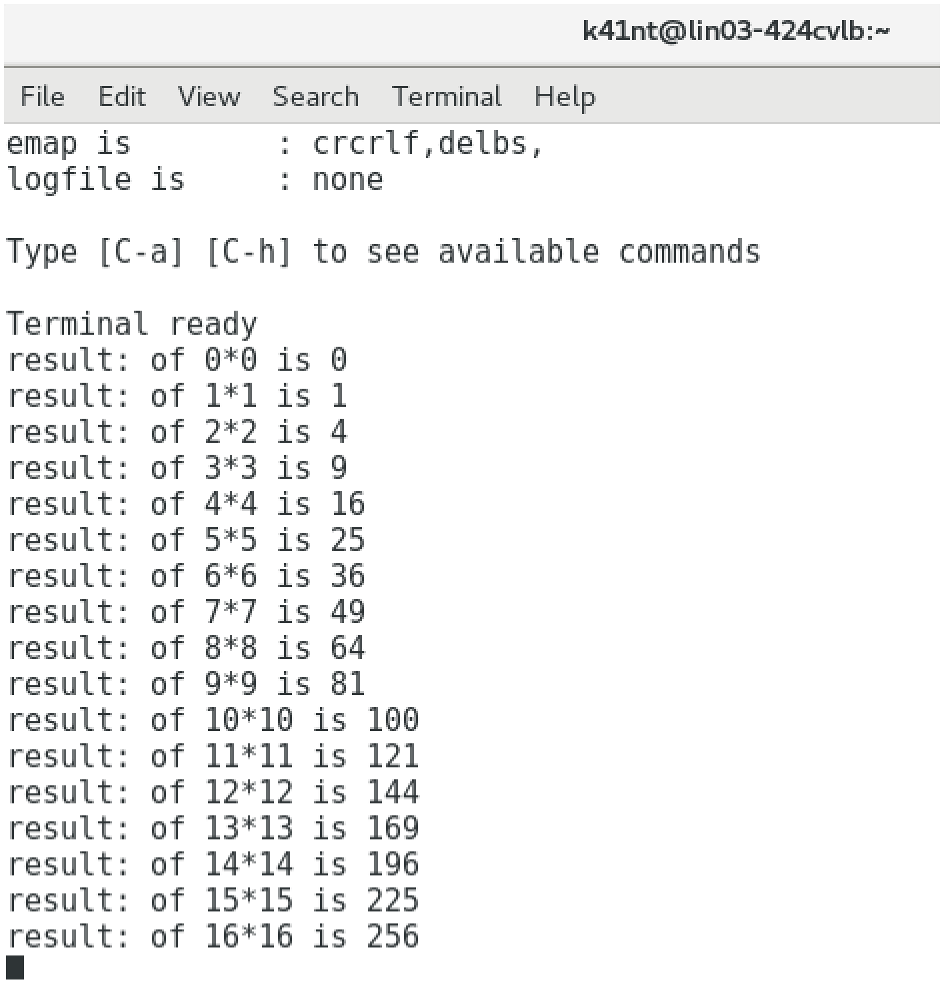
7/ Create and import the provided

8/ Program the FPGA and verify the result my monitoring through picocom.

RESULT:

Check through the header files (xparameters.h and multiply.h) to obtain the instructions for read and write as well as the RegOffset value.

The result must be less than 32bits, since the multiplicands are 32 bit unsigned integer type.



```
emap is        : crcrlf,delbs,
logfile is     : none

Type [C-a] [C-h] to see available commands

Terminal ready
result: of 0*0 is 0
result: of 1*1 is 1
result: of 2*2 is 4
result: of 3*3 is 9
result: of 4*4 is 16
result: of 5*5 is 25
result: of 6*6 is 36
result: of 7*7 is 49
result: of 8*8 is 64
result: of 9*9 is 81
result: of 10*10 is 100
result: of 11*11 is 121
result: of 12*12 is 144
result: of 13*13 is 169
result: of 14*14 is 196
result: of 15*15 is 225
result: of 16*16 is 256
```

Result sreenshot

<u>**CONCLUSION:**</u>

I learned how to create and customize an IP design by creating a peripheral modules and implement it into a Zynq sytem.

The process of reading through the header files and the C code helps me understand the syntax and the use of the new functions in SDK.

I also learned how to use picocom to display the result from the program.

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?

Tmp_reg is used as a flip flop the synchronize the values with the clock. It also helps reduce the risk of error during the process. Errors in the input will be transferred to the tmp_reg. tmp_reg will get to its previous state if the error is detected and therefore, only correct values will get to the output. If the tmp_reg is removed, then the code might have race conditions during the synthesis process which causes race conditions that leads timing errors.

2/ 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.

The program will produce incorrect results when:

- The two input and the output registers are all 32 bit long. The output can easily be overflow if the two inputs are too large.
- The program will not work for signed numbers. It's call the context determine operation, it happens when one variable is declared as signed and the other is not. The result will lose the sign because the program will treat two inputs as unsigned.

Solution for this is declaring both the inputs as signed:

```
//User Defined Code
reg signed [0:C_S_AXI_DATA_WIDTH – 1] tmp_reg, slv_reg2;
// Declaring both variables as signed
input signed [0:C_S_AXI_DATA_WIDTH – 1] slv_reg1, slv_reg0;
always @ (posedge S_AXI_ACLK) begin
        if (S_AXI_ARESETN == 1'b0) begin
                slv_reg2 <= 0;
                tmp_reg <= 0;
        end
        else begin
                tmp_reg <= slv_reg1*slv_reg0;
                slv_reg2 <= tmp_reg;
        end
end
```

C CODE:

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

void print(char *str);

int main()
{
   init_platform();
   int i = 0;
   int out;
   for(i = 0; i<=16; i++){
          MULTIPLY_mWriteReg(XPAR_MULTIPLY_0_S00_AXI_BASEADDR, 0, i);
          MULTIPLY_mWriteReg(XPAR_MULTIPLY_0_S00_AXI_BASEADDR, 4, i);
          out = MULTIPLY_mReadReg(XPAR_MULTIPLY_0_S00_AXI_BASEADDR, 8);
          printf("result: of %d*%d is %d\n\r",i,i,out);
   }



   cleanup_platform();
   return 0;
}
```