ELECTRICAL & COMPUTER ENGINEERING
TEXAS A&M UNIVERSITY

ECEN 449: Microprocessor System Design

Lab 3: Creating a Custom Hardware IP and Interfacing it with Software

Kylan Lewis
UIN: 719001131
ECEN 449-504
TA: Ashwin Ashokan
Date: 10/2/2020

Introduction:

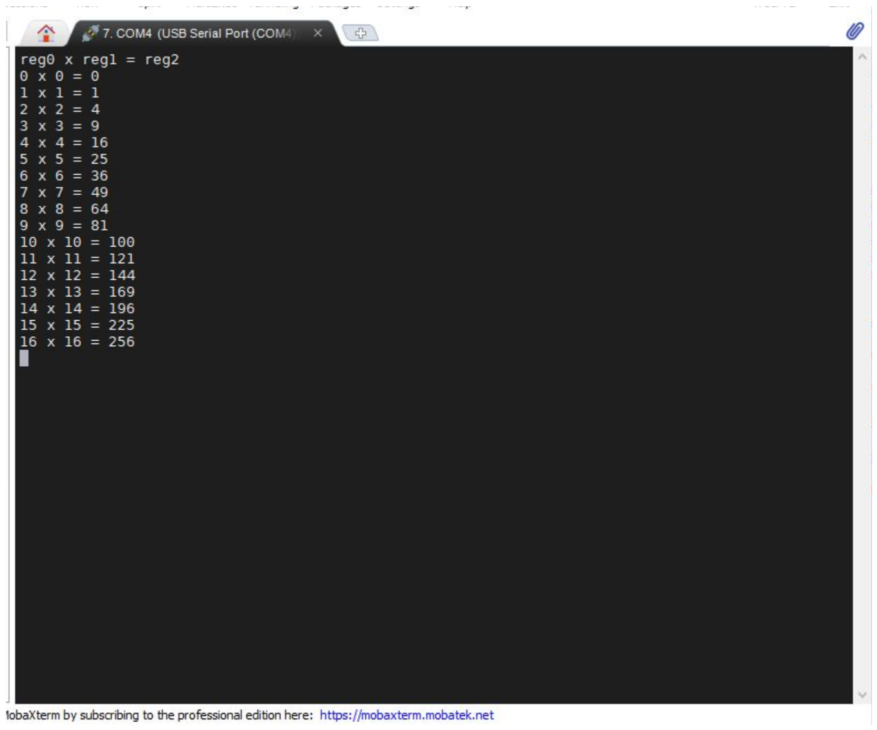
The purpose of this lab is to learn how to do the general procedures of implementing the microprocessors and GPIO using block diagrams in vivado and using the SDK to run code on the FPGA board.

Procedure:

1. Create a new vivado project using the Zybo Z7-10
2. Create a new block diagram called "multiply" using the custom ZYBO_Z7_B2.tcl file
3. Re-customize the IP to the lab manual specifications
4. After finishing, edit "multiply_v1_0_S00_AXI.v" by adding the necessary user logic from the manual and commenting out the writes to the slv_reg2.
5. Run connection Automation and create an HDL wrapper
6. Generate the bitstream and export the hardware after for SDK .
7. Launch SDK to create a new project with the default "helloworld.c" file.
8. Edit the helloworld.c file to write to slv_reg0 and slv_reg1 and read a multiplication results into slv_reg2 and the console.
9. Follow the design hints found in the lab manual to accurately test your functionality.
10. Program the FPGA, run the .elf file on hardware, and use MobaXterm to verify the console results.

Results:

The results of this lab included having a proper console output after working with Verilog and C to understand and generate a functional program. Essentially we used Verilog to develop a block diagram and manage the functionality for the slv0, slv1, and slv2 registers and C was used to write and read data from those registers. The FPGA is programmed with the SDK and runs our multiplication program. We can then see the output over the COM4 dataline and see that the program functions as intended in the screenshot below.


7. COM4 (USB Serial Port (COM4)
lobaXterm by subscribing to the professional edition here: https://mobaxterm.mobatek.net

```
reg0 x reg1 = reg2
0 x 0 = 0
1 x 1 = 1
2 x 2 = 4
3 x 3 = 9
4 x 4 = 16
5 x 5 = 25
6 x 6 = 36
7 x 7 = 49
8 x 8 = 64
9 x 9 = 81
10 x 10 = 100
11 x 11 = 121
12 x 12 = 144
13 x 13 = 169
14 x 14 = 196
15 x 15 = 225
16 x 16 = 256
```

Post-Lab Questions :

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?

The temporary register acts as a delay. If we removed this register from the code, the multiplication block would be inconsistent.

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 multiplication block only supports 32-bit unsigned numbers. If the slv_reg0 and slv_reg1become larger than 32 bits, it would generate incorrect results. The name assigned to this type of computation error is overflow and to correct it we would need to change the width to something larger than 32 bits (64 bits or even larger).

Conclusion:

This lab was very easy to complete. For the most part all we had to do was follow the instructions posted in the lab manual and then program the FPGA afterwards. The only complication that I had was including the right header file after writing MULTIPLY_mWrite/Read functions in the C code. Since I didn't select a ZYBO board from the device section I had to include the header #include <xil_io.h>.

Code:

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


int main()
{
  init_platform();
        // Declare variable to write the result to and print it
    int result;
        // Declare i outside of the parenthesis; used to iterate
    int i;
    for (i = 0; i <= 16; i++) {

       //write to the slv_reg0
       //MULTIPLY_mWriteReg(BaseAddress, RegOffset, Data) BaseAddress = 0x43C00000, Offset =0

       MULTIPLY_mWriteReg(0x43C00000, 0, i);

       //write to the slv_reg1
       //MULTIPLY_mWriteReg(BaseAddress, RegOffset, Data) BaseAddress = 0x43C00000, Offset =4

       MULTIPLY_mWriteReg(XPAR_MULTIPLY_0_S00_AXI_BASEADDR, 4, i);

       //read from the slv_reg2 and console
       //MULTIPLY_mReadReg(BaseAddress, RegOffset) BaseAddress = 0x43C00000, Offset =8

       result = MULTIPLY_mReadReg(XPAR_MULTIPLY_0_S00_AXI_BASEADDR, 8);
       printf("%d x %d = %d\n", i , i, result);

    }

    cleanup_platform();
    return 0;

}
```