ECEN 449 Lab Report 3

Philip Smith - 624002014 (Sec. 511)

February 15, 2020

Introduction

In Lab 3 I got a chance to create my own IP block and integrate it with the PS7 (Processing System 7). The PS7 from Xiling is a System on a Chip (SoC) IP block that contains a dual core processor, meomory controller, several interfacing options such as SPI, I^2C, 1000BaseT Ethernet and more. It allows for connectivity to programmable logic via ARM AXI busses.

Procedure

The first step is to create an IP block with the appropriate amount of registers. In this case we only need 3. Then we write a simple definition in Verilog that multiplies two registers and places them into an output register as follows:

```
tmp reg <= slv reg0 * slv reg1 ;
slv reg2 <= tmp reg;
```

After this we connect it to PS7, synthesize, and export out bitstream to the SDK where we can write a C program to reference it at a particular offset in the memory map.

Results

There were no issues to report and overall this was the easiest lab yet. The program ran fine and displayed the expected values. The following is my serial console output:

Conclusions

This was an interesting lab that raised some questions for me. In the SDK the memory map for my multiply IP block seems to take up 64 kilobytes of address space. This, I feel, is far in excess of what it actually needs. There are only 4 registers involved in the synthesis of this IP block, each at 32 bits wide, meaning I would expect the IP block to take 16 Bytes of address space. The SDK does not specify what is taking up the rest of the block's allocated memory.

Questions

1. The tmp_reg serves as an intermediate register to impart a “synthesizeable” delay into the circuit. It should be noted that, as specified in the Lab 3 instructional video, this is no longer necessary and therefore its removal would have no consequences.

2. Due to the size of the output register, which is 32 bits wide, there exists conditions in which when multiplying two 32 bit values we get a result in excess of 2^{32}, which would cause and *overflow* and subsequently an incorrect result from the IP block. To correct this in the most direct way I would convert the output register to 64 bits wide. Although it would be possible to do this in verilog by changing the value of C_S_AXI_DATA_WIDTH, I would feel more comfortable allowing Vivado to handle this automatically.

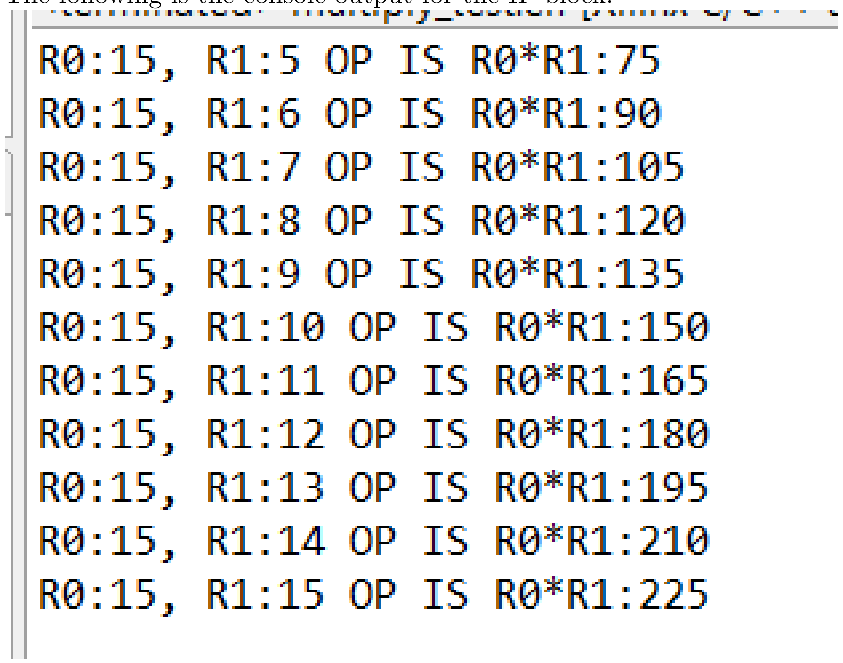
Appendix A - helloworld.c

The following is the code for helloworld.c:

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

//Memory Map of multiply IP
//0x43C00000        r0                  32b
//0x43C00004        r1                  32b
//0x43C00008        r2                  32b
//For whatever reason it needs 65k of memory???

void print(char *str);

int main()
{
    unsigned char i; //Iterator for the loop
    unsigned char max_val = 16; //Iterations to run
    unsigned int result; //Result of IP block

        init_platform();

    print("---------- Multiply Test Program ----------\n\r");
    printf("\t+ multiply: I am at offset %x\n\r", XPAR_MULTIPLY_0_S00_AXI_BASEADDR);
    for (i = 0; i <= max_val; i++){

            MULTIPLY_mWriteReg(XPAR_MULTIPLY_0_S00_AXI_BASEADDR, 0x00, i); //Load first value into reg. 1
            MULTIPLY_mWriteReg(XPAR_MULTIPLY_0_S00_AXI_BASEADDR, 0x04, max_val - i); //Load second value into reg. 2
            result = MULTIPLY_mReadReg(XPAR_MULTIPLY_0_S00_AXI_BASEADDR, 0x08); //Read result from reg 3

            //Print results of operation
            printf("R0: %d\tR1: %d\tR2: %d\tOPERATION IS R0 * R1\n\r", i, max_val-i,result);

    }

    cleanup_platform();
    return 0;
}
```

Appendix B - Console Out

The following is the console output for the IP block:



```
R0:15, R1:5 OP IS R0*R1:75
R0:15, R1:6 OP IS R0*R1:90
R0:15, R1:7 OP IS R0*R1:105
R0:15, R1:8 OP IS R0*R1:120
R0:15, R1:9 OP IS R0*R1:135
R0:15, R1:10 OP IS R0*R1:150
R0:15, R1:11 OP IS R0*R1:165
R0:15, R1:12 OP IS R0*R1:180
R0:15, R1:13 OP IS R0*R1:195
R0:15, R1:14 OP IS R0*R1:210
R0:15, R1:15 OP IS R0*R1:225
```