

ECEN 749: Microprocessor Systems Design

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

Manav Gurumoorthy

830000011

Section 603

Table of Contents

Introduction.....	3
Procedure	4
Configuration	4
SDK	4
Results.....	5
C Code	5
Conclusion	7
Questions	8

Introduction

In this lab I wrote code to run on the Zynq microprocessor on the Zybo board. In contrast to lab 2 which used the MicroBlaze soft processor, which was defined on the FPGA, this lab uses the Zynq microprocessor which is a physical processor present on the Zybo board. A custom IP was created that performed multiplication of two numbers, this was created as a peripheral to the processor and was interfaced using UART.

Procedure

Configuration

The initial configuration is like what was done in lab 2 to create the block diagram for the design. One change in this lab is the use of the Zynq microprocessor IP instead of the MicroBlaze IP. Once the Zynq block was added, this IP was reconfigured. A configuration file (.tcl) was used to setup the processor.

All interfaces in the processor are disabled, except the UART 1 interface, this is the interface used to interact with the custom IP block. Once the processor was setup, the design of the custom IP begins.

The custom IP block was a simple block that multiplies two numbers stored in a register and puts the result out onto a third register. The IP block was created as an AXI4 Peripheral device. After the setup of the device the Verilog code of the IP was modified to include a user defined module that performs the desired product operation. The multiply code was added to the multiply_v1_0_S00_AXI.v file, also all write instruction to slave register 2- the output register- was commented out in order to prevent any erroneous writes to this register. The Verilog code was saved and checked for errors, then the IP was packaged.

The custom IP block was then available to be added to the design. The block was added, and automated connection was run. The setup of the block design was now complete.

The HDL wrapper (top module) was created for the block design. Upon successful generation of the wrapper the bitstream was generated. There are many warnings that pop-up during this process, but these can be ignored as they don't affect the task.

Once the bitstream was generated, the hardware with the bitstream was exported.

SDK

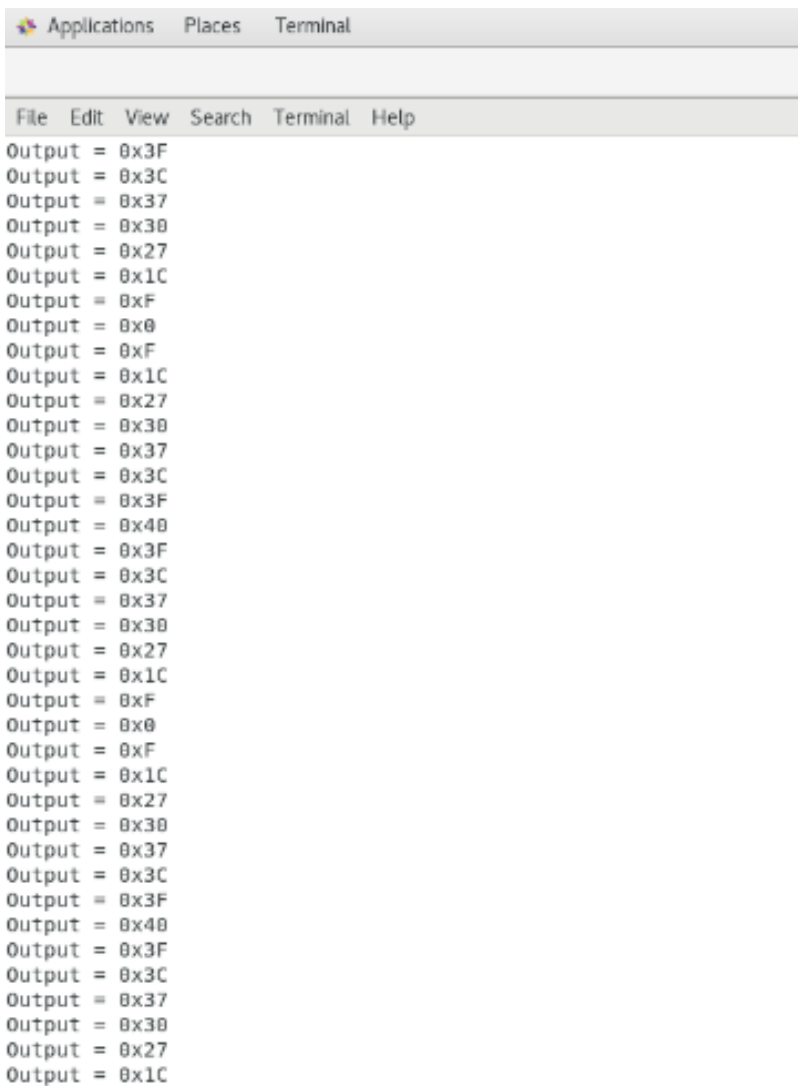
After the design was configured and the bitstream was generated, the SDK was launched to develop the application to write to and read from the registers to be used for multiplication in the processor. The code was written in 'C' and was created in a helloWorld template. The SDK already added some header files and created a basic structure for the code. To be able to access the APIs to our Verilog module, the SDK created a header file called 'mutiply.h', this file was included in the code along with 'xparameters.h'. The two files held APIs to perform read-write to the registers and details about the address and offset of these registers respectively.

Using the right function calls a C application was created to multiply two numbers. The code for the same can be found at the end of this document.

The results of the multiplication was visible on the console monitor, which used xil_printf, hence the xil_printf.h header was also included. The output results was sent to the console monitor on the USB/tty hub at a baud rate of 115200, using this information, the results were viewed over 'picocom'.

Results

The results of the multiplication were viewed using picocom, a screenshot of the same is attached below.



```
Applications  Places  Terminal
File Edit View Search Terminal Help
Output = 0x3F
Output = 0x3C
Output = 0x37
Output = 0x30
Output = 0x27
Output = 0x1C
Output = 0xF
Output = 0x0
Output = 0xF
Output = 0x1C
Output = 0x27
Output = 0x30
Output = 0x37
Output = 0x3C
Output = 0x3F
Output = 0x40
Output = 0x3F
Output = 0x3C
Output = 0x37
Output = 0x30
Output = 0x27
Output = 0x1C
Output = 0xF
Output = 0x0
Output = 0xF
Output = 0x1C
Output = 0x27
Output = 0x30
Output = 0x37
Output = 0x3C
Output = 0x3F
Output = 0x40
Output = 0x3F
Output = 0x3C
Output = 0x37
Output = 0x30
Output = 0x27
Output = 0x1C
```

The program has a for loop that generates two numbers for multiplication, one number counts from 0 to 16 and the other number counts down from 16 to 0, the picocom terminal is shown above.

C Code

```
/*
 * helloworld.c: simple test application
 *
 * This application configures UART 16550 to baud rate 9600.
 * PS7 UART (Zynq) is not initialized by this application, since
 * bootrom/bsp configures it to baud rate 115200
 *
 * -----
 * | UART TYPE   BAUD RATE |
 * -----
 *   uartns550   9600
 *   uartlite    Configurable only in HW design
 *   ps7_uart    115200 (configured by bootrom/bsp)
 */

#include <stdio.h>
#include "platform.h"
#include <multiply.h>
#include <xparameters.h>
#include <xil_printf.h>
//void print(char *str);
```

```

int main()
{
    u32 temp,temp2,output;
    int i =0;
    init_platform();
    temp = 0;
    temp2 = 0;
    while(1){
        //xil_printf("Hello World\n\r");
        // = MULTIPLY_mReadReg(XPAR_MULTIPLY_0_S00_AXI_BASEADDR,MULTIPLY_S00_AXI_SLV_REG0_OFFSET);
        for (i = 0; i<16; i++){
            temp = i;
            temp2 = 16-i;
            MULTIPLY_mWriteReg(XPAR_MULTIPLY_0_S00_AXI_BASEADDR,MULTIPLY_S00_AXI_SLV_REG0_OFFSET,temp)
;
            MULTIPLY_mWriteReg(XPAR_MULTIPLY_0_S00_AXI_BASEADDR,MULTIPLY_S00_AXI_SLV_REG1_OFFSET,temp2);
            output = MULTIPLY_mReadReg(XPAR_MULTIPLY_0_S00_AXI_BASEADDR,
MULTIPLY_S00_AXI_SLV_REG2_OFFSET);
            xil_printf("\n\rOutput = %d", output);
        }
        cleanup_platform();
    }
    return 0;
}

```

Conclusion

Lab 3 introduced the Zynq processor present on the Zybo board. I also created a custom IP that multiplies two numbers, it was interfaced with the software running on the Zynq processor.

Questions

1. The temporary register holds the value of the product of the two operands, and then this value is written to the output register.
2. One computation error that can occur during multiplication is overflow, i.e. the multiplication of two large numbers and the output is too large to fit in the output register.

```
module multiply (a,b, out) ;  
  
    input reg [32:0] a,b;  
    reg [32:0] temp;  
    output reg [32:0] out;  
    temp = a * b;  
    if (a == 0 || b == 0) out = temp; // checking for overflow  
    else if ( a == temp / b) out = temp; // checking for overflow  
    else  
        out = 32'bxxxxxx;  
  
endmodule
```

If an overflow occurs, the output is set to 'x'. Overflow is checked as follows, if both the numbers are zero then no overflow occurs, also if one operand is equal to the product divided by the other operand , no overflow has occurred, else overflow has said to have occurred.