CSE 260 – Digital Computers: Organization and Logical Design

Lab 5

Doug Shook *Due 12/03/2015*

Introduction

In this lab you will be tasked with designing and implementing a simple 3-bit CPU. This CPU will consist of an ALU and a register file.

1. Design

In lab 2, we began piecing together an ALU (though you may not have realized that's what we were doing at the time). In this lab, we are going to extend the functionality of the ALU to incorporate some additional operations, as well as the ability to store results to registers. The result will be a CPU.

In order to control our simple CPU we are going to need an *instruction set* and instruction set is a list of acceptable operations that the CPU can perform, as well as a way to input the addresses of registers needed to perform these operations. Our instructions will be 12 bit values. The first 3 bit will describe an operation, the next three bits will be an address of a source register, the following 3 bits will be the address of a destination register, and the final 3 bits will an input value. A table describing our instruction set is given below:

Instruction (first three bits)	Operation
000 - nop	Do nothing
001 - load	reg[destination] = dataIn
010 - increment	reg[destination] = reg[destination]++
011 – decrement	reg[destination] = reg[destination]--
100 – shift left	reg[destination] = shift left(reg[source])
101 – shift right	reg[destination] = shift right(reg[source])
110 – addition	reg[destination] = reg[source] + dataIn
111 – subtraction	reg[destination] = reg[source] - dataIn

So for example, an instruction that looks like 110101001010 is asking us to compute the following:

reg[1] = reg[5] + 2

Notice that not all instructions require the entire 12 bits to be functional. This means you will need to process the instructions differently depending on the operation that is taking place.

You will also need to construct a second, separate module (please keep them separate!) that manages the register file. This register file should be able to store 8 3-bit values, so it is an 8x3 register file. This configuration allows us to keep the computation part of the CPU separate from the storage part of the CPU. Please review Zybook section 6.3 and the accompanying video for more information about how register files work and how they can be implemented in VHDL.

You can use your lab 2 code as a starting point for your CPU module (or you can start from scratch if you would like). The register file module should be instantiated inside the CPU module, and the CPU module is responsible for controlling the register file to perform all of the reads and writes correctly.

You must draw a circuit that represents your CPU module. You may use a component symbol like the one shown in the Zybook for the register file – you do not need to show how the register file works in your circuit.

2. Implementation

Create a new project and module and implement your modules using VHDL. Some things to note:

* All data values should be assumed to be 3 bit unsigned integers. You do not need to worry about overflows.

* You are not required to explicitly implement states in VHDL, but you may if you wish.

* You may have multiple read ports for your register file, but you may only have one write port.

* In addition to the inputs that the CPU needs to function (clk, reset, instruction), you should include an additional input called regAddr. This input will be 3 bits and should be used as an address into your register file. The value of the register at address *regAddr* should be output from your CPU module. We will use this to debug your CPU on the FPGA and ensure that it is working correctly.

* To perform integer casting, you will probably want to add the commonDefs.vhd code to your project (see the zip file below), then add the following statement at the top of your register file module:

```
use work.commonDefs.all;
```

3. Testing

Once you have finished writing code for your FSM, you need to create a testbench for it. Your testbench needs to test every possible operation. There are 8 operations total. Once you are satisfied with your testbench and simulation results, you should print out a copy of your simulation. You should then annotate your simulation printout with each operation that takes place.

4. Hardware Implementation

After successfully simulating and testing your circuit, you should implement it on an FPGA. This means you must complete a top.vhd module that connects your module to the lcdDisplay, buttons, knob, and LEDs on the FPGA. Download these files and add them to your project:

http://cse.wustl.edu/~dshook/cse260/labs/lab5.zip

The only file you should modify is top.vhd. You need to do the following:

* Instantiate your CPU component.

* Create and connect signals so that the instruction to the CPU is controlled by the knob and displayed on the top row of the LCD display.

* Connect a button that controls when an instruction is read. If the button is pressed, the instruction should be passed to the CPU. When the button is not pressed, the CPU should do nothing.

* Connect a button to the reset signal. This reset signal should clear out your registers and reset the CPU.

* Connect the *regAddr* input to the switches. You can use swt(3 downto 0) to reference the switches.

* Connect the output of the CPU to the bottom row of the LCD display.

Once you have finished modifying top.vhd, you should deploy your project on the FPGA just as you have done on previous labs. You should see the input on the top row of the lcdDisplay, controlled by the knob as in the past. You should be able to use the switches to examine the content of one of the registers and verify the operation of the CPU. Note that if you push the knob down on the FPGA, it should allow you to change the instruction value faster.

What to Turn In

You must turn in a hard copy of the following:

* Your CPU module circuit.

* An annotated printout of your simulation, as described above.

* A completed demo form (see below).

You must turn in a digital copy of the following (on Blackboard):

* Your VHDL code for the CPU and register modules.

* Your VHDL code for the testbench.

* Your completed top.vhd module.

Demo

You must demo your lab on an FPGA before you turn it in. Print out a copy of the demo form from the course website and hand it to the TA. They will walk you through the demo, fill out the form, and then give it back to you. Staple the completed form to the front of your report when you hand it in.