CSC258H Week 6 Lab: Register Files

1 Introduction

This week, we're working with larger-scale devices composed of multiple individual components. The goal of the lab is to become more comfortable with high-level design – *programming* with hardware devices, rather than mathematical specification using gates. In this lab, you'll also explore the use of control bits in devices.

Before lab, read through the lab handout so that you are familiar with the procedure. Prepare schematics for the required register and register file, using higher-level devices like muxes and D flip-flops whenever possible. In particular, this lab requires the D flip-flops you created in week 5 as well as multiplexers and demultiplexers. For the multiplexers and demultiplexers, you can either (a) use your multiplexer from week 4 and create a demultiplexer or (b) use the LPM_MUX and LPM_DECODE library elements. I recommend the latter. This tutorial describes how to insert an module from the library:

`https://www.altera.com/en_US/pdfs/literature/tt/tt_my_first_fpga.pdf`

Warning: This lab has, in the past, been the longest lab of the semester, since it requires that you combine multiple high level components into a new device. That requires some creativity and time spent problem solving. The prelab work asks that you complete a rough design so that the TAs can review it to make sure that you have a correct design for implementation. To get full credit for the lab, you may wish to spend time before lab wiring the parallel register. You may also need to spend time after the lab finishing the testing of the register file component. It will be used in next week's CPU lab.

2 Parallel Registers

A register is a device that stores a value. In reality, registers (and register files) are often created out of SRAM cells rather than flip-flops, but for simplicity, we'll use D flip-flops as our storage element. You should already have a 1-bit D flip-flop from last week. To create an n-bit parallel register, wire n D flip-flops in parallel; each flip-flop stores one bit.

Create a register that stores 4-bit values. Your register should take a 4-bit data input and produce a 4-bit data output. (Use buses to create the input and output. A bus is just a multibit wire. You can get a single bit from the bus by using array notation. If you have a bus A[1..0], you can name an input A[0], for example, to get bit 0 from the bus.) Your register should also take *enable* and *clock* inputs. *enable* is an example of a "control" input – it determines whether or not the register is to accept input.

When *enable* and the *clock* both go high, your flip-flop should accept input. If either is low, your flip-flop should maintain its current state. (Recall that the flip-flop is an edge-triggered device, so the input value is stored on the positive edge, when the second of the two required inputs goes high.) This functionality can be implemented without modifying the internal structure of your D flip-flop: your register design should consist of inputs and outputs, D flip-flop symbols, and *a single basic logic gate to support enable functionality.* Once it is implemented, verify that the enable functionality works using a simulation and create a new symbol for the device. **Show your circuit and simulation to your TA, and be prepared to answer a few questions about how it works.**

3 Register Files

A single register can only hold a single value, and a computer will need more storage. (Most processors have 32 or more.) The *register file* that you will build for the lab is an array of registers that allows one register

to be read and one register to be written at any time. It will be built using your mux and register designs. You will also find it useful to use a demultiplexer/decoder (you may use one from the Quartus library or create one).

Your register file will contain 4 registers (referenced by the numbers 0-3). Your register file should accept a 4-bit data input and produce a 4-bit data output. It also requires a *clock*, *enable*, and *two* 2-bit *select* inputs. All four of the latter inputs are "control" inputs. The first *select* is a *read_select*: the value of the input indicates which of the four registers is the source of the register file's output. The second *select* is a *write_select*: the value of the input determines which of the four registers will accept input in the next clock cycle. Like the register design, the *clock* and *enable* inputs determine if and when a value is to be stored.

Here are two questions to consider as you build your register file:

- All four registers will be providing output at the same time. How do you *choose one* to be the output for the register file? What width must the devices that chooses have?
- Only one register should *write* its value at any given time. Here are two options, *only one of which is correct*: you could use a 1-bit demultiplexer (demux) to send the enable bit to only one register, or you could use 4-bit demux to send the input to only one register. *Hint: Remember that a demux sends the input value to the selected output line. The other output lines will have value* 0.

Before lab, sketch out your design on paper. **Show your TA your schematic before you begin construction of this device.** Once your TA and you agree that you have a working design, implement your circuit. You do not need to complete this circuit to get credit for the lab, but you will need it to get credit for next week, so you should finish it this week.

4 FPGA Testing

If time allows, load your register file onto the FPGA. Connect the data input to switches 0-3 (PIN_N25, PIN_N26, PIN_P25, PIN_AE14), enable to switch 4 (PIN_AF14), read-select to switches 5-6 (PIN_AD13, PIN_AC13), and write-select to switches 7-8 (PIN_C13, PIN_B13). Connect the clock to switch 9 (PIN_A13). Connect the data output to four red LEDs (PIN_AE23, PIN_AF23, PIN_AB21, PIN_AC22). This will allow you to load binary numbers into the registers. When you set switch 19, a clock pulse is initiated that should load the value set on switches 0-3 to the register specified on switches 7-8 if and only if switch 4 allows it. You can verify which data values are stored by changing the read-select input.

As you work with your circuit, consider: is *read_select* synchronized with the clock? Should it be? Is it possible to store a value in the middle of a "clock pulse"? Is it possible to store more than one value in a "clock pulse"? What would you need to do to fix those issues?

5 Summary of TODOs

Below is a short summary of the steps to be completed for this lab:

1. Before the lab, read through the lab handout and prepare circuit diagrams for (a) the 4-bit register and (b) register file. Bring these items to lab on paper for your to verify. Use high-level devices like muxes and flip-flops wherever possible.
2. During lab, implement the parallel register and test it. Show the design and test results to your TA. Be prepared to explain how the device works. Save the register as a symbol for use in the register file.
3. Get your design for a register file verified by the TA and then begin working on this device. You will need it for next week, so you may need to spend additional time preparing the register file after the lab is complete. Save your register file as a symbol for next week's lab.

6 Evaluation

3 marks in total: 1 mark for completing and testing the parallel register device *and demonstrating your understanding of the device* to the TA; 1 mark for showing your TA a schematic for a register file that you both agree will work; and 1 mark for working throughout the lab to complete your register file device (or finishing the device, if you worked on it before lab). You must be able to explain your circuits and waveforms to the TA to earn credit.