

CSC258H Lab 6: Register Files

1 Introduction

This week, we’re working with larger-scale devices composed of multiple individual components. The goal of the lab, similar to Lab 3, 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, and the use of buses. A bus is simply a wire containing multiple bits.

Required Submissions: This lab requires the submission of a circuit file (“`lab6.circ`”) and a lab report (“`lab6report.pdf`”) to MarkUs by **Tuesday, March 9, 10:00 PM**. Please read the syllabus regarding the late policy. All submitted work must be completed **individually**.

2 Multi-Bit (Parallel) Registers

A register is a device that stores a value. In reality, registers (and register files) are often created out of SRAM cells, but for simplicity, you can imagine it being built out of D flip-flops. A 4-bit register stores 4 bits in “parallel” (you can imagine each D flip-flop handling one bit of the input and output). We will not be creating this component, and instead will be using the built-in **Register**, but if you feel motivated you can try building it from scratch for fun.

In addition to the clock input, registers take in a **write enable** (WE) input. It is an example of a “control” input – it determines whether or not the register is to accept input. When **write enable** is high, and the **clock** goes high, the register stores the input. If either is low, the register maintains its current state. (Recall that the flip-flop is an edge-triggered device, so the input value is stored on the positive edge of the clock.)

3 Design and Implement a 4-Register 4-Bit Register File

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 from (at any time) and one register to be written to (on the positive edge of the clock). It will be built using Logisim-Evolution’s built in **Multiplexer**, **Demultiplexer** and **Register** blocks.

Your register file will contain 4 registers (referenced by the numbers 0 to 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. Like a register, 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:

- **Q1:** All four registers will be providing output at the same time. How do you *choose one* to be the output for the whole register file? What are the bit-widths of inputs of the device that chooses?
- **Q2:** Only one register should store the input value on the positive edge of the clock. Here are two options, **only one of which is correct**: you could use a 1-bit demux to send the enable bit to only one register, or you could use a 4-bit demux to send the input to only one register. Which option should we choose and why? *Hint: Remember that a demux sends the input value to the selected output line. The other output lines will have the value 0.*

Before the lab, prepare schematics for the register file using high level blocks. In your lab report, include the circuit diagram for the register file. Be sure to indicate which wires and inputs are buses and their “width” (number of bits).

At the beginning of lab, your TA will take questions about the register file designs. Once you’re confident in your design, ask your TA to take a look before implementing your device. Once you believe you have a working design, implement your circuit.

You need to use multi-bit components in your design. Refer to the Logisim-Evolution reference section 3 for more help, but the relevant properties are **Data Bits** and **Select Bits** to customize the components.

4 Lab Report

Include the following in your lab report named “`lab6report.pdf`”.

1. Your name and student number.
2. A screenshot of the circuit diagram of the register file implemented in Logisim.
3. A short paragraph discussing the answer to the question Q1 in Section 3.
4. A short paragraph discussing the answer to the question Q2 in Section 3.

5 Summary of TODOs

1. Before the lab, read through the lab handout. Complete the design of the circuit diagram.
2. In the lab, discuss your design with your TA then implement the register file.
3. Once convinced the register file works, demonstrate it to your TA.
4. Complete the lab report.
5. Submit `lab6.circ` and `lab6report.pdf` to MarkUs before the deadline.

Evaluation (2 marks in total):

- 1 mark for the circuit diagram in the lab report.
- 1 mark for the answers to Q1 and Q2.

For the next and final Logisim-Evolution lab, we'll combine a register file with a simple instruction decoder and ALU, to build the datapath and control unit for a simple processor that you can program yourself!