Lab 9: Building a Register File

CS/CoE 0447(B) Spring 2015 — Dr Kosiyatrakul (Tan)

Released: Wednesday, November 11, 2015
Due: Monday, November 16, 2015

Submission timestamps will be checked and enforced strictly by CourseWeb; **late submissions will not be accepted.** Remember that, per the course syllabus, if you are not marked by your recitation instructor as having attended a recitation, your score will be cut in half. **Follow all instructions.**

A register file is a subcircuit that lets the surrounding circuitry read from one or two registers, which are chosen dynamically. A register file also lets the surrounding circuitry optionally write to one register, which is also chosen dynamically. In this lab, you will build a register file that has two read ports and one write port. That means that your register file circuit should let outside circuitry read two registers[1] at a time while also supporting writing to one register at a time. **You should support eight registers total. Each register must hold 32 bits of data.** Use Logisim's built-in register component.

Figure 1 shows a symbolic representation of what your register file subcircuit should looks like. The slash indicates that an input/output contains more than one bit of data.

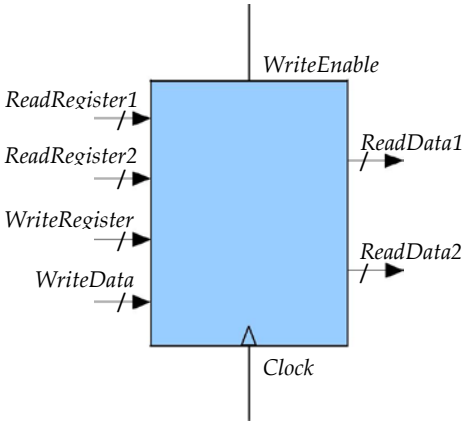

Figure 1. *A symbolic representation of a typical register file.*

- ***ReadRegister1*** specifies which register should be read from. That register's data should be presented on the *ReadData1* output.
- ***ReadRegister2*** specifies which register should be read from. That register's data should be presented on the *ReadData2* output.
- ***WriteRegister*** specifies which register should be written to.
- ***WriteData*** is the data that will be written to the register specified by *WriteRegister*. On a rising clock edge, the data will be written to the specified register.
- ***WriteEnable,*** if true, will cause the *WriteData* to be written to the register specified by *WriteRegister*; otherwise, *WriteData* will be ignored and no register will be written to.
- ***Clock*** conveys the clock signal.

Before starting, ask yourself the following questions:

1. How can we tell how many bits will be outputted on *ReadData1*? On *ReadData2*?
2. How can we tell how many bits specify *ReadRegister1*? *ReadRegister2*?
3. How can we tell how many bits specify *WriteRegister*?
4. How can we tell how many bits specify *WriteData*?

[1] The registers read do not necessarily have to be different. For example, it is valid to read from $r0 and $r1, but it is also valid to read from $r2 and $r2 simultaneously.

5. How can we tell how many bits specify *WriteEnable*?
6. Why is *WriteEnable* necessary?
7. Why does the circuit not have *ReadEnable* inputs?
8. Why does the register file have a clock input? Why doesn't it use its own clock instead of requiring a clock from outside circuitry?

To assist you, here are some tips about sizing different bit widths:

1. Since the registers are 32 bits wide, and since *ReadData1* will output the entire contents of a register, *ReadData1* will be 32 bits wide. By the same reasoning, *ReadData2* will be 32 bits wide.
2. Since you need 8 registers, then 3 bits are needed to "name" each register. The register having the name 000 is register 0, 001 is register 1, 010 is register 2, and so on. *ReadRegister1* and *ReadRegister2* are therefore each 3 bits wide.
3. WriteRegister describes which register of the four registers is being written to. Therefore, *WriteRegister* is 3 bits wide.
4. *WriteData* specifies the data that will be written into a register. Since the registers are 32 bits, *WriteData* will be 32 bits.
5. *WriteEnable* says whether or not a register write should actually be performed (yes or no). Therefore, *WriteEnable* is 1 bit wide.

Here are some general tips to help you implement the register file:

- The data that should be written to a register must be able to be "given" to *any* of the 8 registers in the register file (*i.e.*, your circuit must be able to write to any register).
- Registers can be made read-only by setting one of their pins appropriately. This can let you control which register, if any, is written to. Take a look at the Library Reference for more information.
- Consider where you might need multiplexers or demultiplexers:
  - Recall that, given multiple inputs, a **multiplexer** lets you select from exactly one of them. Remember that the number of inputs from which a multiplexer chooses can be configured by changing its "select bits" property, and that the "data bits" property controls how many bits are on each of its inputs.
  - A **decoder** has multiple output wires, but only one of those output wires is true at any given time. It is easy to control which output is true. Decoders are found under the plexers category. A **demultiplexer** is like a decoder in that it sends its input to exactly one of many outputs (the other outputs will be set to 0).

You can poke a register and give it a value. This will let you give each register a different value to make sure that the circuitry reads from the correct register (according to *ReadRegister1* and *ReadRegister2*).

To help you get started, Figure 2 on the next page shows a partially built register file with four 16-bit registers.

▶ **Save your completed Logisim file as `lab09.circ` and submit it via CourseWeb.**

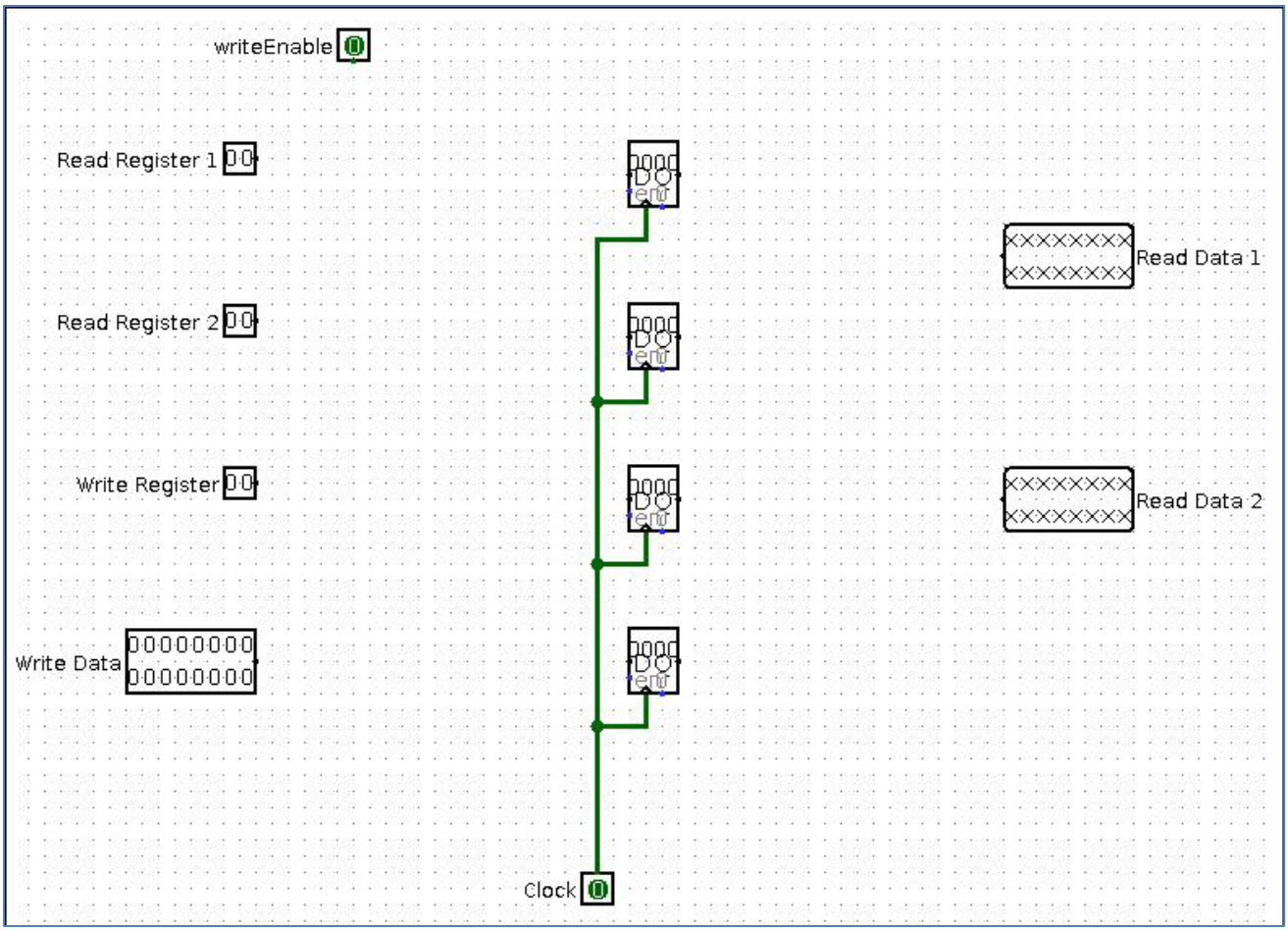

Figure 2. *A not-yet-completed register file containing four 16-bit registers. In this example, the outside circuitry is requesting: a) that register 0 is read from and output on ReadData1, b) that register 0 is read from and output on ReadData2, c) that no register is written to (writeEnable is 0). If writeEnable was 1, then register 0 should become all 0s on the next clock tick rising edge while the other registers should keep their value.*