6.2: Multiplexors, Adders, and Registers

Topics:

- Multiplexors
- Multi-bit components and connections
- Adders
- Registers

Introduction:

In Lab 6.1, we looked at logic gates, and building simple logic components using truth tables. In CSc 256, we’ll mostly be studying and using high-level logic components that are pre-built. This lab will cover some simple high-level components and their behavior.

The Logisim examples covered in class and for this lab can be found at:

[http://unixlab.sfsu.edu/~wusu/csc256/LABS/DOCS/LogisimExamples/](http://unixlab.sfsu.edu/~wusu/csc256/LABS/DOCS/LogisimExamples/)

Steps:

In lecture, we looked at a multiplexor, a component that lets us select between two inputs. In a 2-input multiplexor, with two inputs 0 and 1, a 1-bit select control indicates whether the output comes from input0 or input1. Open the Logisim file 4_MuxAdder.circ. This file contains three independent circuits; the top one is a 2-input (or 2 x 1) multiplexor, with 1-bit data.

![Multiplexor Diagram](image)

Multiplexors are often draw with a somewhat different shape, like this:
We should get used to looking at both ways of drawing multiplexors.

Back to the Logisim circuit… The select control is now set to 0, so the output of the multiplexor comes from input0. Click on the Poke tool (the icon on the top left toolbar), then clock on input0 and input1 to change them. We’ll see that the output follows input0, but ignores input1.

Now change the select control to 1. Change input0 and input1; we’ll see that the output follows input1!

Usually we don’t just have one data bit for inputs and outputs of multiplexors; it’s more common to have many data bits connected in a circuit. The middle circuit in 4_MuxAdder.circ shows a 2-input multiplexor with 8 data bits in each input and output. This still allows us to select from input0 or input1, using the select control (still one bit, of course); however, input0, input1, and output are all 8 bits wide:

Again, a 2 x 1 multiplexor with 8-bit data is often drawn like this (the / 8 means the wire is 8-bit wide):
Back to the Logisim circuit... Currently, the select control is set to 0. Again, use the Poke tool to change bits in input0 and input1. We'll see that the output follows input0 only, and ignores input1. When we change the select control to 1, and change input0 and input1, we'll see that the output follows input1 only.

In Lab 6.1, we looked at building a 2-bit adder, i.e., a circuit that has two inputs, both 2-bit integers, and adding them together to produce a 3-bit output. Let's look at an 8-bit adder; this is the bottom circuit in 4_MuxAdder.circ:

Click on the Poke tool, and click on bits in input0 and input1. We'll see that the 8-bit sum appears at the output. (If you're interested in exploring further, the cout bit is the extra carry-out bit that may be produced by the addition.)

In lecture, we looked at building a 4-bit register out of D flip-flops. Let's look at a bigger 8-bit register in 5_8BitReg.circ:
Note that the symbol for a register is just a D flip-flop! (That’s essentially what a register is.) Other than the data input and output (both 8 bits here), there are two additional control inputs. The enable signal controls whether or not the input is stored in the register. When enable is 0, the input is ignored, and the contents of the register do not change; when enable is 1, the input will eventually be checked and stored in the register.

Again, in your textbook, an 8-bit register is usually drawn like this:

So what did we mean by “eventually” the input will get stored into the register? As we discussed in class, the state of the register (i.e., what the register contains) doesn’t change until it sees a rising edge of the clock. The clock goes from 0 to 1 to 0 to 1 repeatedly. It’s only at a transition from 0 to 1 that the input is stored into the register, and appears at the output.

Go back to the Logisim register. Initially, the enable control is set to 0. Use the Poke tool to change the input of the register. You can make the clock tick by clicking repeatedly on the clock icon. Note that the register does not change, even though the clock is ticking, because the enable control is set to 0.

Now set the enable control to 1. Click on the clock icon. Now we see the input get stored into the
register, and appear on the output pins.

Note that the Logisim clock component is basically the same as a 1-bit input pin! In lectures, we mostly used a 1-bit input as a clock, and clicked it repeatedly to get the clock transitions. Either approach is fine, of course.

Now we’re ready to look at a more elaborate circuit with a register and other components. Let’s look at a simple version of a circuit that implements some of the functionality of the Program Counter (PC) in a MIPS-like CPU. For simplicity, in our version, our mini PC will contain 8 bits, not the full 32 bits.

What does our mini PC have to do? When our MIPS program starts running, we have to place the address of the label main into the PC. Hence, we’ll need 8-bit input pins going into an 8-bit register, and 8-bit output pins so we can see what the register contains. Then we add an enable signal and a clock signal:
So this is a basic 8-bit register; not very exciting. Since it’s the PC, we’ll need to be able to add 4 to the contents, to get to the next instruction. We’ll connect the output of the mini PC to one input of an 8-bit adder. The other input of the 8-bit adder will have input pins connected, which we’ll eventually set to 4:
Now we need to connect the output of the 8-bit adder back into the input of the mini PC. We can’t do this directly of course; the register input pins are already directly connected to it. (Remember from lecture that an input cannot have two different sets of signals connected to it.) So we need a multiplexor at the input of the mini PC, to select between the register input pins, and the output of the adder. So here’s our final mini PC circuit; in our current design, multiplexor select 0 connects PC + 4 to PC, select 1 connects the register input pins to the PC.

You don’t have to build this circuit; it’s already in 6_Plus4.circ !

Let’s see this in action. We have to initialize a few values.

Obviously, the constant 4 input needs to be initialized to 4. (Logisim also provides a constant component, under the Gates menu, that will work here. In a real circuit, the appropriate bits would be connected directly to +5V or 0V.) Use the Poke tool to set the constant 4 input pins to 4 (0000 0100).

It would also be appropriate to have the mini PC start at some contents other than zero. Suppose in our 8-bit CPU, all programs start execution at address 0x50, or 0101 0000. Use the Poke tool to set the register input pins to 0101 0000.

To write 0101 0000 into the PC, we need the multiplexor to connect input 1. So set the select 0/1 pin to 1.
In addition, we need to set the *enable* control for our mini PC to 1.

Note that through all these steps, our mini PC has not changed its contents. Recall that a register does not change its contents until its enable signal is set, and it sees a clock transition from 0 to 1.

Now we’re ready to start the clock! Initially, the clock input is 0 when we open the Logisim circuit. When we click on the clock pin once, it goes to 1, and we see that our mini PC gets its new contents: 0101 0000.

(You might want to click the clock pin again, so it goes back to 0.)

So we’ve initialized our mini PC. Let’s try to add 4 to it. Note that the output of the mini PC is always connected to the adder; hence, the output of the adder is *always* 4 + the mini PC. All we have to do is select it through the multiplexor, make sure the *enable* pin for the mini PC is set to 1, then send a rising edge (0 to 1) of the clock to the mini PC. These again are the steps:

Set *select 0/1* to 0.

Make sure *enable* is 1, and *clock* is 0.

Click on clock; it goes from 0 to 1, and the mini PC has 4 added to it; it goes from 0101 0000 to 0101 0100!

Give the clock a few more clicks, to add 4 several times to the mini PC. This is very similar to what the MIPS PC does when we run a program. An instruction is processed, then the PC gets incremented by 4, so it points to the next instruction. If there’s a branch instruction, the PC has to change to the branch target address. This would be similar to entering a new address on the *register input* pins, setting *enable* to 1, and giving the clock a rising edge.

We’ll revisit this PC increment logic in Chapter 7, and also look in detail at other components that make up a simple MIPS CPU.

The last Logisim component we’ll look at is a Read Only Memory (ROM). This is a simple memory block, with address pins and data pins, in 7_ROM.circ.
Currently, the ROM has 8 address bits and 8 data bits; each location is a byte. Byte 0 is initialized to 0x1a, byte 1 to 0x2b, byte 2 to 0x3c, and so on. Try changing the memory address pins; the output pins will show the contents of the byte at that address.

We’ll use some of these components in the Lab 6 exercises, and also in Chapter 7.