Lab 6: Introduction to Logisim

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

Released: Wednesday 21 October 2015
Due: Monday 26 October 2015, 11:59pm

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.**

In this lab, you will experiment with **Logisim** and build a sample circuit. Logisim is a tool for designing and simulating logic circuits. It's surprisingly powerful, free (as in *libre*), runs on Java, and is available online from `http://www.cburch.com/logisim/`

A. Let's build an adder!

First let's learn a bit about Logisim:

- Square boxes are **input pins.** Their values are being used to compute a value.
- Circle boxes are **output pins.** Their values are being computed by the circuit and output to circuitry that wants the result of the one-bit addition.
- Triangles with circles are **inverters** or **NOT-gates**, as they invert whatever their input is.
- Lines are **wires.** **Bright green wires** are "on" (true, or 1). **Dull green wires** are "off" (false, or 0).
- **Blue wires** are unconnected. We often have blue lines (e.g., when a 5-input AND-gate only has 2 inputs connected, the blue inputs will be ignored). If you get blue lines as outputs when you're not expecting them, though, it's probably a good idea to restart Logisim.
- **You never want to have red wires.** These indicate an error, such as two outputs being connected together.

All wiring components are available from the main tool bar (below the "File" menu). Note that, by default, Logisim creates logical gates that can accept up to five inputs. For a more realistic circuit, **you should set the number-of-inputs property to 2.** For example, each AND-gate you create should have exactly two inputs. Two-input gates most closely represent what is available when creating real circuits and let us better understand things like propagation delay and the need to simplify circuits.

Also, be sure to enter a **label** for each input and output pin in its attribute table. The attribute table is near the lower-left-hand corner of Logisim's window.

There are two main modes of operation in Logisim:

- The **poke tool** (the hand icon) lets you change the values of input pins to test different inputs.
- The **edit tool** (the arrow icon) lets you add, select, and manipulate wires.

Here are a few more useful tips when building circuits:

- Clicking and dragging creates **wires.** The point where you start the click and where you let go of the click determines where the wire connects to other things. If, for example, you click then drag the mouse over another wire, the two wires will not be joined. If you click to draw a wire, then let go of the mouse while it is positioned over top of an existing wire, then the new wire and existing wire will be joined. Large dots show where wires connect to other wires.
- You can "undraw" wires to shorten them, or use the **Delete** key to remove the selected wire.
- To move segments of wires, you may find **Alt-Click** useful. For complicated circuits, this can help you clean up a messy design. But if you're not careful, it can also complicate a simple one! As usual, undo is your friend and is just a **Ctrl-Z** away.
- You can easily **rotate** circuit components. For example, when you add an AND-gate to a circuit, it faces "east" by default (that is, the input comes from the left side and the output is on the right side). Clicking on the AND-gate lets you change some of its properties. Changing the "facing" property rotates the gate. You can also click on a gate and use the arrow keys to rotate it.
- **Optical illusion:** Sometimes, a wire looks like it is connected to a component, but it really is not. Zoom-in on the component to ensure that the wire is connected to the component's pin. The zoom control is in the lower-left-hand corner of Logisim's window.
- Multi-bit input and output pins are easily confused. To check a pin component, examine the **output** property in the pin component's attribute table. You may also notice that input pins have square corners and output pins have rounded corners.
- Gates have a **size** property. Left click an existing gate and change its size. Since the narrow size is usually sufficiently large, this can free up space in your circuit.
- Gates also have a **number-of-inputs** property to set how many inputs they can use. Again, this should typically be set to 2.
- If you get an "oscillation apparent" error, it indicates that you are in some way creating an invalid "loop" of wires. For example, the output of a gate may indirectly loop back onto the same gate, causing the gate to keep switching (oscillating) between outputting 1 or 0. You should never get this error if you are doing things properly.
- If you are still having an unexplained problem after considering the tips above, save your file, and then try to resolve the problem by restarting Logisim.

Now that you know all about the basic functionality of Logisim, now let's consider building a one-bit adder. Recall that a one-bit adder has **three inputs:** the first one-bit number you are adding, the second one-bit number you are adding, and a carry-in bit. There are **two outputs:** the answer (result of the addition) and the carry-out bit. Several one-bit adders can be chained together to build an n-bit adder (where n is as large as desired).

Your task is to construct a one-bit adder circuit based on the circuit design shown on the next page.

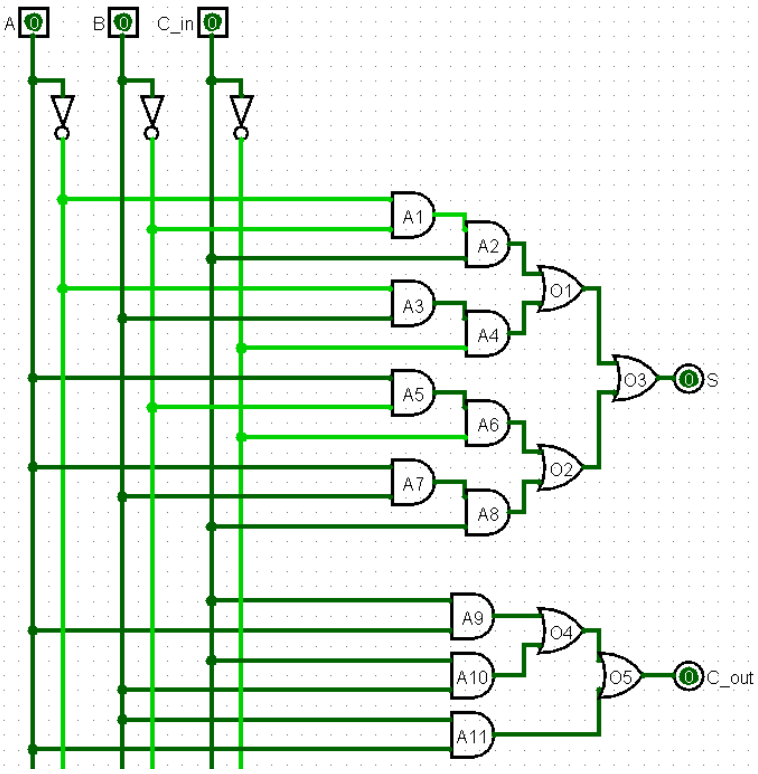

Figure 1. *A one-bit adder circuit. Here, 0 + 0 + 0 is being added, to produce 0 (sum) and 0 (carry out).*

▶ **Save your circuit as `lab06part1.circ`.**

B. Let's build an adder-subtractor!

While the one-bit adder we built is undoubtedly beautiful, Logisim also lets us leverage many **useful abstractions** which make our circuits more understandable, more compact, and easier to design.

Remember how we said that n one-bit adders, chained together, can be used to form an n-bit adder? Well, Logisim has a built-in adder component that we can use under the "Arithmetic" section. We can simply specify the number of bits, n, in the attribute window, and place an adder block into our circuit. It's as easy as that.

But how do we provide multibit inputs to this adder block? We can add input and output pins to our circuit, just as before, but we'll change the "data width" (in bits) to match the number of bits the adder supports and is expecting. When we draw wires coming out from these multibit inputs, we notice that they are **black wires.** In reality, these black lines represent "bundles" of wires, specifically one wire to transmit each bit of the value being conveyed. But Logisim lets us handle that by drawing one line!

So, if we have an 8-bit bundle coming from an input into an 8-bit adder, everything's great and hooked up as expected! However, if you get **orange wires** and you receive an "Incompatible widths" error, this means that your data widths don't match somewhere. Look for the little orange numbers that appear at the circled pins to find out what data widths you have so that you can adjust them in the attribute window to match.

So **let's walk through** building a simple eight-bit adder-subtractor which, given two numbers *A* and *B*, can compute either *A* + *B* or *A* – *B*. The circuit is shown in the figure below:

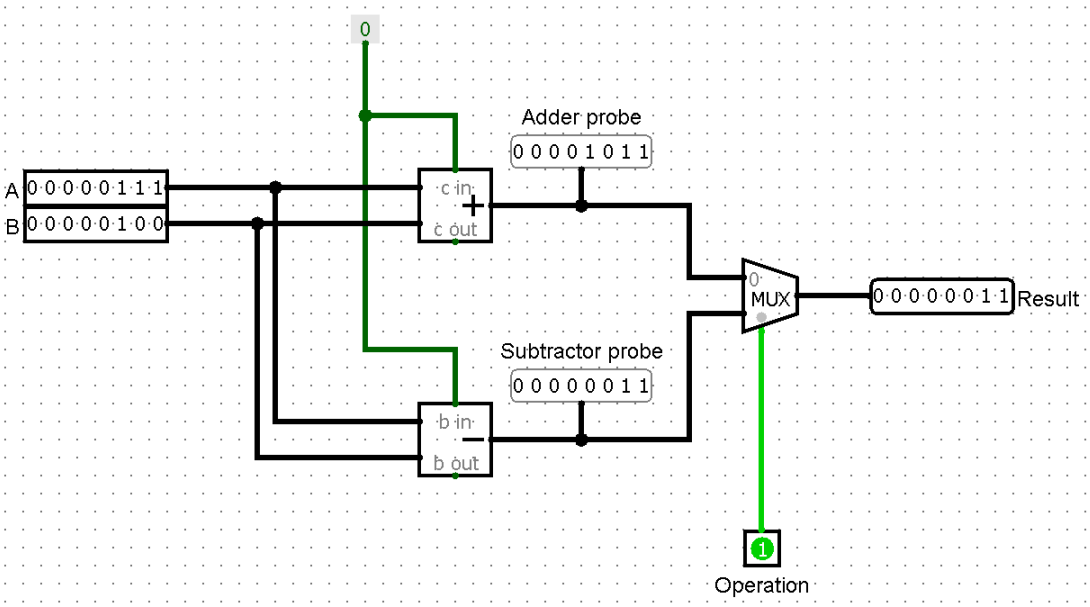

Figure 2. *A basic eight-bit adder-subtractor (minimal ALU), implemented using a multiplexer to select the operation. Here, the computation 7 – 4 = 3 is being performed.*

First, let's populate our circuit with the stuff we already know about:

- An 8-bit input for the first number, *A*
- An 8-bit input for the second number, *B*
- A 1-bit input to select the operation (addition vs subtraction)
- An 8-bit output for the result
- An 8-bit adder (found under "Arithmetic")
- An 8-bit subtractor (also found under "Arithmetic")

Start by connecting the inputs, *A* and *B*, to the input pins of the adder and subtractor as in the figure above. **How can we be sure we're connecting the wires as expected?** When using the edit tool (the arrow icon), not only can we draw wires, we can also mouse over the input and output pins of blocks like the adder and subtractor to learn more about their purpose.

For example, while addition is commutative, so it doesn't matter whether we connect *A* or *B* to the adder's first input pin as long as we connect the other to the second one, the same is not true for subtraction. *A* – *B* is a different calculation than *B* – *A*. So if we mouse over the top-left input pin of the subtractor, we see that that is the pin for the minuend (the number from which to subtract). Since we want to be able to compute *A* – *B*, this would be *A*. Likewise, we can discover that the lower-left input pin is for the subtrahend, *B*. **We can also find extensive information about any built-in circuit block under "Help > Library Reference."**

Now, we can piece the rest together with the addition of a few more components from the component window:

- **Wiring > Constant.** In addition to the two inputs *A* and *B*, our adder takes a single bit as "carry in", and our subtractor takes a single bit as "borrow in". For our calculations, we want these values to always be zero, so we'll use a Constant with a data width of **1 bit** and a **value of 0,** and connect it accordingly.
- **Wiring > Probe.** One of the disadvantages of the "black-wire" multibit bundles Logisim gives us is that we can't see the values transmitted on those wires as easily as we could with the single-bit scheme of light and dark greens. While you can use the poke tool to inspect these bundles one by one, sometimes we'd like to have a permanent "window" into the values they carry, and that's exactly what we get when we attach a probe. Attach one to the result of your adder and one to the result of your subtractor.
- **Plexers > Multiplexer.** Also known as a "mux," a multiplexer is used to select one input from many and pass it along as its output. Since we have the results of both $A + B$ and $A - B$ computed, we want to use a mux to choose between them for our final output. Since we have two inputs to choose from, and it takes only one bit to enumerate those choices, this means we need **1 select bit.** Since the data width on each input is 8 bits, the mux must support **8 data bits.** Now, we can connect each "potential result" to one of the mux inputs, and connect our operation selection input to the select pin of the mux. (**NOTE:** We will also generally always set the "Include Enable?" attribute of muxes to "No", both for style and clarity, so that we don't confuse our select signal with the enable signal that is sometimes used with muxes.)

That's it! You should now have a working eight-bit adder-subtractor, which is effectively a minimal version of the arithmetic-logical unit (ALU) you'd see in a real processor! Use the **poke tool** to poke your inputs, changing the numbers around to see how everything interacts within your circuit.

- ▶ **Save your circuit as `lab06part2.circ`.**
- ▶ **Zip your two circuits for this assignment into a single zip file called `lab06.zip` and submit it via CourseWeb.**