CS 0447 — Spring 2017

Lab 8: Introduction to Logisim

Released: Thursday 2 March 2017, 11:00am
Due: Monday 20 March 2017, 11:59pm

Each of you should submit your own solution via CourseWeb. Each person must turn in their own copy of the lab. If you choose to work with a neighbor/partner, put your partner's name on your submitted copy of the lab. Submission timestamps will be checked, and late submissions will not be accepted. Be sure to name your files appropriately as indicated in the 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 from **http://www.cburch.com/logisim/**

IMPORTANT: You must only use two-input gates. For example, each AND gate you create should have exactly two inputs. You will lose points if you do not follow this rule. 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. By default, Logisim creates gates that can accept up to five inputs. Be sure to only "use" two of those inputs (i.e., connect only two things up to each gate's inputs).

Part 1 of 3: Let's Build an Adder

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 joined together to build an n-bit adder (where n is as large as desired).

Start Logisim and create a new circuit. **Save this file as "lab08part1-<your Pitt username>.circ".** In this file, you will create a one-bit adder, but 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 input will be ignored).
- **You never want to have red wires.** These indicate an error, such as two outputs being connected together.

All components are available from the main tool bar (below the "File" menu).

The poke tool (the hand icon) lets you change the values of input pins to test different inputs. The arrow tool lets you add, select, and manipulate wires. You can "undraw" wires to shorten them, or use the delete key to remove the selected wire.

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

Here are a few 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. Green dots show where wires connect to other wires.
- 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 up/down/left/right 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 (e.g., change the size to narrow). This can free up space in your circuit.
- Gates also have a **number-of-inputs** property to set how many inputs they can use.
- 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, create a one-bit adder, basing your work on the circuit design shown on the next page. Make sure the names of the input and output pins are the same as in the circuit below.

Submit your circuit to CourseWeb. Name your file "**lab08part1-<your Pitt Email ID>.circ".** For example, "**lab08part-xyz123.circ**" (Instead of your PeopleSoft number, use your Pitt Email ID).

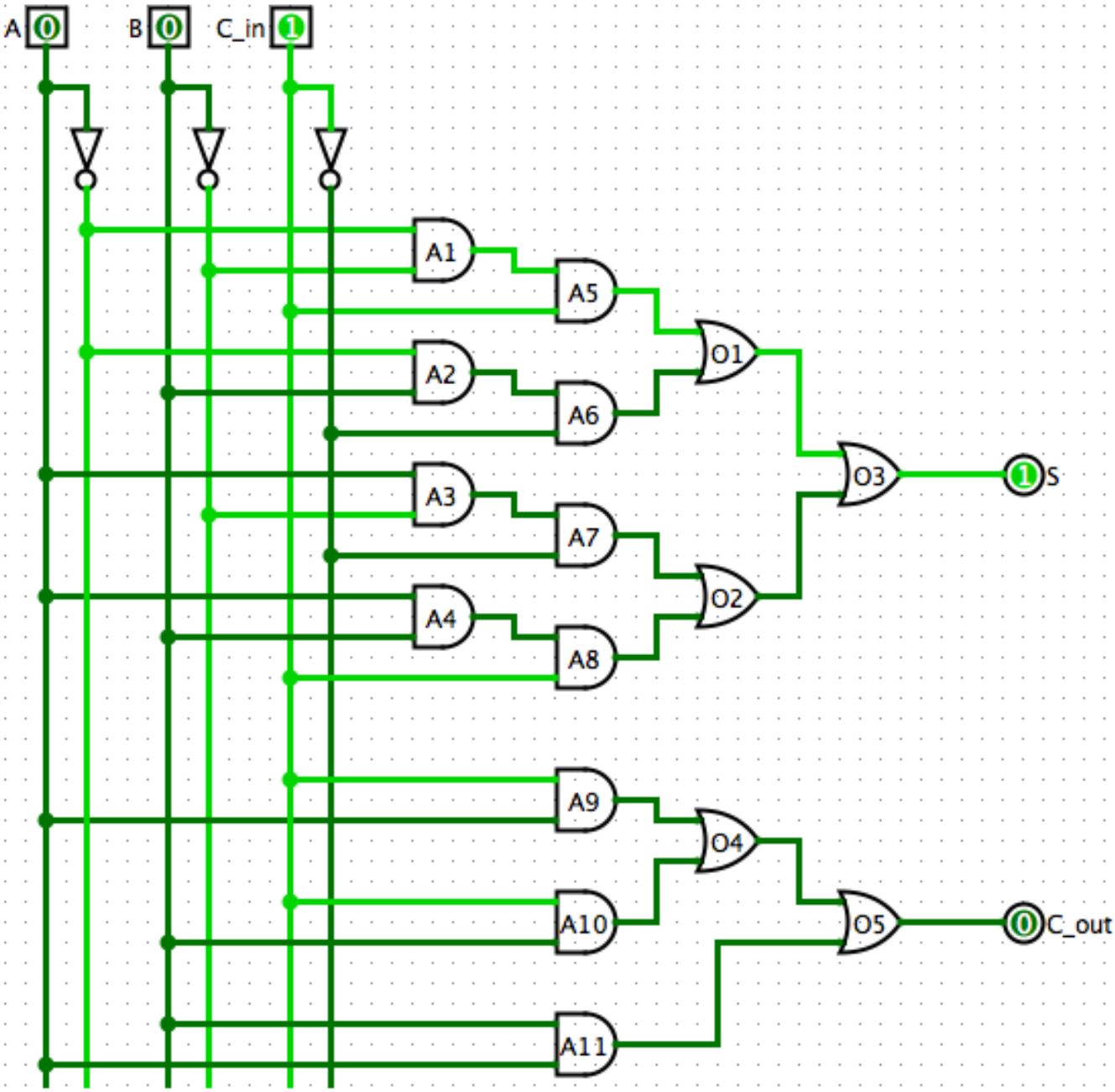

Figure 1. *A one-bit adder circuit. Here, 0 + 0 + 1 is being added, to produce 1 (sum) and 0 (carry out).*

Part 2 of 3: Adder Analysis

The circuit that you built in Part 1 implements the following Boolean equations, which are the Boolean equations for a one-bit adder:

$$S = \overline{AB}C_{\text{in}} + \overline{A}B\overline{C_{\text{in}}} + A\overline{BC_{\text{in}}} + ABC_{\text{in}}$$

$$C_{out} = AC_{\text{in}} + BC_{\text{in}} + AB$$

For each combination of inputs A, B, and C_{in}, determine the outputs of each AND gate **A1–A11,** each OR gate **O1–O5,** and the final outputs **S** and $\mathbf{C_{out}}$**,** according to the diagram on the previous page. By changing the values of the input pins, observe the behavior of your circuit and fill out the following **truth table**. A truth table describes the behavior of a circuit given all possible inputs.

Using only zeroes and ones, **enter your answers from the table below into CourseWeb.**

A	**B**	**C_{in}**	A1	A2	A3	A4	A5	A6	A7	A8	O1	O2	O3	A9	A10	A11	O4	O5	**S**	**C_{out}**
0	**0**	**0**																		
0	**0**	**1**																		
0	**1**	**0**																		
0	**1**	**1**																		
1	**0**	**0**																		
1	**0**	**1**																		
1	**1**	**0**																		
1	**1**	**1**																		

Part 3 of 3: Building a Tester

Let's consider a system that determines something very important, for instance a computer that determines the current altitude of an aircraft. If this computer gives the wrong answer, it can have detrimental results. So, in such cases we sometimes simply use three computers and rely on the answer if two of them give the same result. However, in this case, we are going to hypothetically give them a problem we know the answer to and we want to detect a failure (when two or more report 0)

In this part, you will build such a circuit for a 3-input tester that outputs a 1 whenever the test fails: that is, when the number of 0's in the inputs is at least 2. (In our hypothetical situation, we are assuming the correct answer is 1, so if we see two answer 0, we know something is wrong) The non-simplified equation for this tester is:

$$V = \overline{A}BC + A\overline{BC} + \bar{A}B\bar{C} + \overline{ABC}$$

Part a: Truth Table / Derivation

As an exercise for yourself, you may fill out this truth table to see how you would derive the non-simplified equation: (**Note: you don't need to turn in anything for this part**)

A	B	C	V	Explanation
0	0	0	1	**The number of 0s (3) is two or more**
0	1	0		
0	0	1	1	**The number of 0s (2) is two or more**
0	1	1	0	**The number of 0s (1) is less than two**
1	0	0		
1	1	0		
1	0	1		
1	1	1		

And then notice that V in the equation is selecting each row in the table where V is 1 and writing out A, B, and C as they are described on that row. When one is 0, there is a bar placed above it to indicate that it should be considered logically negated. V = $\overline{A}BC$ means V = (not A) and (not B) and C, which is true ***only*** when A and B are 0 and C is 1 (see the highlighted row in the table above)

Part b: Logisim

Once you understand this equation and how it is derived, build a circuit computing the above Boolean formula in Logisim. Your circuit should have 3 inputs labeled A, B and C and 1 output labeled V.

Submit your circuit to CourseWeb. Name your file "**lab08part3-xxx123.circ**" (where xxx123 is your Pitt username, not your PeopleSoft number)