Lab 1: Intro to Logisim

8/25/2022

1 Possible Point

Offline Score: N/A

Add Comment

8/23/2022

Details

Task: In this lab, you will take your first steps with the Logisim software as well as build your very first circuit(s).

Collaboration Policy: As with all labs, you are welcome to get help from fellow students or any 3410 staff member. However, the purpose of this lab is to familiarize yourself with Logisim (which you will need to use for Projects 1 & 2), so we want each of you working on your own circuit. *If you have questions, please ask them in your Lab Section!*

Due date: This lab, like all labs, has no formal deliverables. The point is for you to develop your skills in a stress-free environment. Do your best! That said, **attendance will be taken** and is incorporated into your semester grade. Labs will always appear to be "due" on Thursdays at 11:59PM. In reality, you must attend whenever it is that your lab meets. But in all cases, this lab time falls between the Wednesday and Friday lecture. Thursday is just a simplifying approximation.

For this lab, there are 3 designated **Checkoffs**.

Welcome to Logisim! Logisim is a logic simulator that allows you to design and simulate digital circuits using a graphical user interface (GUI). Logisim comes with libraries containing basic gates, memory chips, multiplexers and decoders, and other simple components. Later in the semester, you will use these components to build a RISC-V (pronounced "risk-five") processor. In this lab, we will guide you through the process of creating an adder in Logisim.

Task 0: Obtaining Logisim

In this step, you'll be downloading Logisim on your personal computer. If you're attending the in-person lab, you can use the lab computer instead--keep in mind, however, that you will need to have it on your personal computer for future assignments.

1. Logisim requires Java 16 or later. If you do not already have it on your computer (or have an older version of Java), please follow the installation guides below:
   - MacOS: **https://docs.oracle.com/en/java/javase/16/install/installation-jdk-macos.html#GUID-C5F0BF25-3487-4F33-9275-7000C8E1C58C (https://docs.oracle.com/en/java/javase/16/install/installation-jdk-macos.html#GUID-C5F0BF25-3487-4F33-9275-7000C8E1C58C)**
   - Windows: **https://docs.oracle.com/en/java/javase/16/install/installation-jdk-microsoft-windows-platforms.html#GUID-A7E27B90-A28D-4237-9383-A58B416071CA (https://docs.oracle.com/en/java/javase/16/install/installation-jdk-microsoft-windows-platforms.html#GUID-A7E27B90-A28D-4237-9383-A58B416071CA)**
   - Linux: **https://docs.oracle.com/en/java/javase/16/install/installation-jdk-linux-platforms.html#GUID-737A84E4-2EFF-4D38-8E60-3E29D1B884B8 (https://docs.oracle.com/en/java/javase/16/install/installation-jdk-linux-platforms.html#GUID-737A84E4-2EFF-4D38-8E60-3E29D1B884B8)**
2. Download Logisim from **Electronic Resources (https://canvas.cornell.edu/courses/42905/pages/electronic-resources)**. This is a Cornell snapshot of Logisim Evolution. It comes as a jar file that will work on any operating system with java installed. Use this snapshot so that we are all using the same version of Logisim for the entire semester. Make sure the

(https://canvas.cornell.edu/courses/42905/modules/items/1511779)

(https://canvas.cornell.edu/courses/42905/modules/item

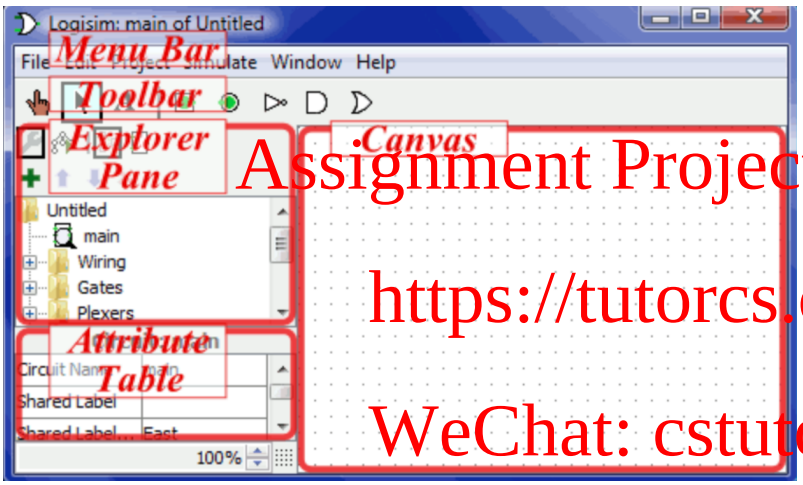
3. To execute the program: on Windows and OS X systems, you will be able to start Logisim by double-clicking the jar file. If that doesn't work, or if you use Linux or Solaris, you can type: **java -jar big-red-logisim.jar** at the command line. (Note: If you are on OS X, the first time you might get a popup suggest the program cannot be opened because it's from an untrusted source. If you do not find the option to "Open" from there, go to "Security & Privacy" under settings for your computer, and you should be able to "Open Anyway" from under the General tab)

Task 1: The Beginner's Guide

Logisim is a simple tool; most of the features you will need are well documented in the reference document. Some main ones you'll come to know (and hopefully love) are **Analysis Circuit**, **Test Vectors**, and the newly added **Autosave** feature (special to the Cornell version of Logisim). Looking for more in-depth information, try checking out the guide from the **Logisim Documentation (http://www.cburch.com/logisim/docs.html)** page. Feeling ready? Let's work through a one bit XOR circuit together! (Adapted from **Logisim 2.7.x Beginner's Tutorial (http://www.cburch.com/logisim/docs/2.7/en/html/guide/tutorial/index.html)**)

Step 1: Familiarizing Yourself

Note: Logisim will look a little different for you but, the major points highlighted below are still the exact same



For this initial part, we will only need to really care about the Toolbar and the Canvas. Before we start building, let's try to first conceptualize our XOR circuit with a truth table. A truth table shows how a logic circuit's output responds to various combinations of inputs. An XOR circuit's truth table would look something like:

Inputs		Outputs
A	**B**	**S**
0	0	0
0	1	1

(https://canvas.cornell.edu/courses/42905/modules/items/1511779)
(https://canvas.cornell.edu/courses/42905/modules/item

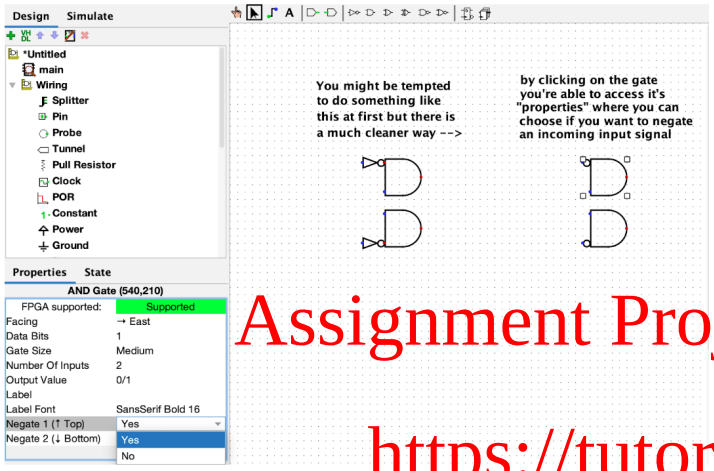
1	1	0

In which, we call our two 1 bit inputs “A” and “B” and our output signal “S”.

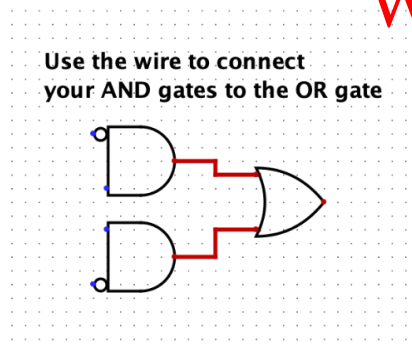
Step 2: The First Circuit

Mini-challenge! Take a few minutes to yourself and try to think about how you’d implement this (if you already have some basic circuit knowledge). Hint: You only need **And**, **Or** and **Not** gates for this!

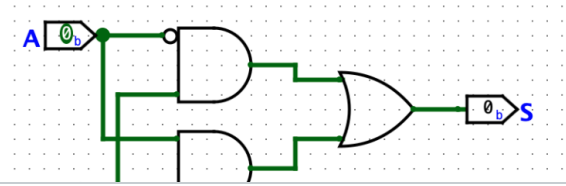
Ready to see if you got it right? Looking at the truth table, I know I need some way to make sure that I output a 1 (or an on signal) if only one of A and B is also a 1. To do so, I’ll create two **And** gates, one which takes A and the negation of B, and vice versa for the other. (You can hover over the icons on the toolbar if you’re unsure about what one of them does)



Now that we can tell when only either A or B is on, we can then use an **Or** gate as our final gate to check if either condition is satisfied from our **And** gates.



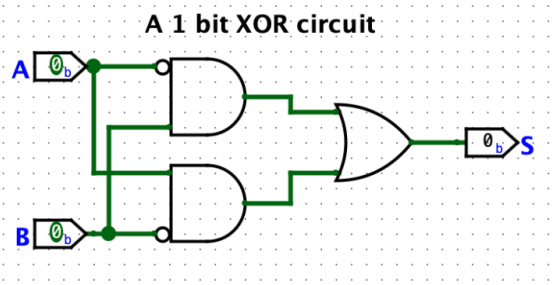
See anything we’re missing? Of course, the input and output pins! Let’s add those. Remember they are **different** pins. You can differentiate them by the location of their wire connect point, if the connect point is on the triangular side, it’s an input pin, and if it’s on the flat side, it is an output pin. If you’re unsure as to how to name your pins, go to the properties tab after clicking on a pin and change its “label” to the name you desire or simply double click on the pin itself.



(https://canvas.cornell.edu/courses/42905/modules/items/1511779)
(https://canvas.cornell.edu/courses/42905/modules/item

Step 3: Finishing Touches

A good practice once you've finished any circuit is to spend some time cleaning up your circuitry (on behalf of all TAs, please) and, depending on the complexity of the circuit, adding some text to help those reading your circuit understand what it is you're trying to accomplish with it (again, on behalf of all TAs, please…), for example:



There you have it, your very first circuit! Not so bad, right?

Step 4: Save It or Lose It

As mentioned earlier in the lab, our version of Logisim comes with an autosave feature, **BUT**, this only goes into effect after you have manually saved it once and given it a file name. Don't fret though, Logisim will give you a pop-up reminding you if you haven't saved your file yet, asking if you're sure you want to close the Logisim window. For practice though, take some time to save your XOR circuit with a proper file name (maybe lab1.circ?).

Note: the auto-save feature is on a set **interval** not based on if a change has occurred or not. This means you **should** still get into the good habit of occasionally saving your circuit manually using Ctrl (Cmd) + S.

Checkoff #1: Show your XOR Circuit to your TA

For more on good design practices for circuits, make sure to check out the **Logisim Design Guidelines (https://canvas.cornell.edu/courses/42905/pages/logisim-design-guidelines)**

Task 2: Add4

Now that you have implemented your first XOR gate in Logisim, let's work on a more complicated circuit -- a 4-bit adder. This will help you get more familiar with the software environment that you will be working in for this class in the next few weeks.

Definition (from **Wikipedia (http://en.wikipedia.org/wiki/Adder_(electronics))**):

A **full adder** adds binary numbers and accounts for values carried in as well as out. A **one-bit full adder** adds three one-bit numbers, often written as A, B, and Cin; A and B are the operands, and Cin is a bit carried in (in theory from a past addition). The full-adder is usually a component in a cascade of adders, which add 8, 16, 32, etc. binary numbers.

In the notation below, **A[4]** denotes that the input is named "**A**" and is 4 bits wide. The input *should not* be named "**A[4]**". Do not create four separate inputs, each named **A** or some variant. The correct approach is to create a *single* input and make it 4 bits wide by changing the "Data Bits" attribute of the input. The attributes appear on the lower left pane of the Logisim window when the input is selected.

Add4:	**S = A + B + Cin; Cout = overflow**
Inputs:	**A[4], B[4], Cin**
Outputs:	**S[4], Cout**

(https://canvas.cornell.edu/courses/42905/modules/items/1511779) (https://canvas.cornell.edu/courses/42905/modules/item

Sub-circuits

Logisim sub-circuits are basically like functions in programming: you make them once, and then you can use them multiple times in a larger context. We're going to use sub-circuits to make wiring easier by building a 1-bit adder, then putting them together into a 4-bit adder, and then eventually (but not today!) a 32-bit adder.

To create a new circuit, select "Project -> Add Circuit..." from the toolbar. To use a circuit (A) as a sub-circuit of another (B), double-click on circuit B on the left pane of Logisim (the *Explorer Pane*). A magnifying glass appears on circuit B's icon, and now the contents of circuit B appear on the right pane (the *Canvas*). Click once on circuit A in the Explorer Pane, then click anywhere in the canvas to place an instance of circuit A as a sub-circuit of circuit B. As one would expect, any time circuit A is updated, all instances of it appearing in circuit B will change their operation in the same way.

For more information on sub-circuits, the **corresponding documentation pages (http://www.cburch.com/logisim/docs/2.6.0/en/guide/subcirc/index.html)** are useful.

**Remember to name the inputs/outputs of any circuits you make by double-clicking on the respective input. This is best practice, can save you the headache of trying to figure out which input is which when using sub-circuits and also becomes important in future projects when we use code in conjunction with our Logisim circuits.

Build a 1-bit adder

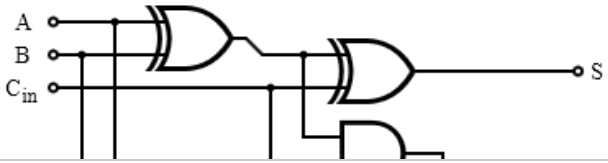
Remember the truth table from earlier? Let's try using it for something a little bit more complex. We've filled in the first line to help you complete the truth table below:

Inputs			Outputs	
A	**B**	**Cin**	**Cout**	**S**
0	0	0	0	0
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

The truth table above can also be expressed using boolean algebra.

S = A xor B xor Cin
Cout = (A and B) or (Cin and (A xor B))

Create the following circuit in Logisim, then save it as an appropriately-named circuit.

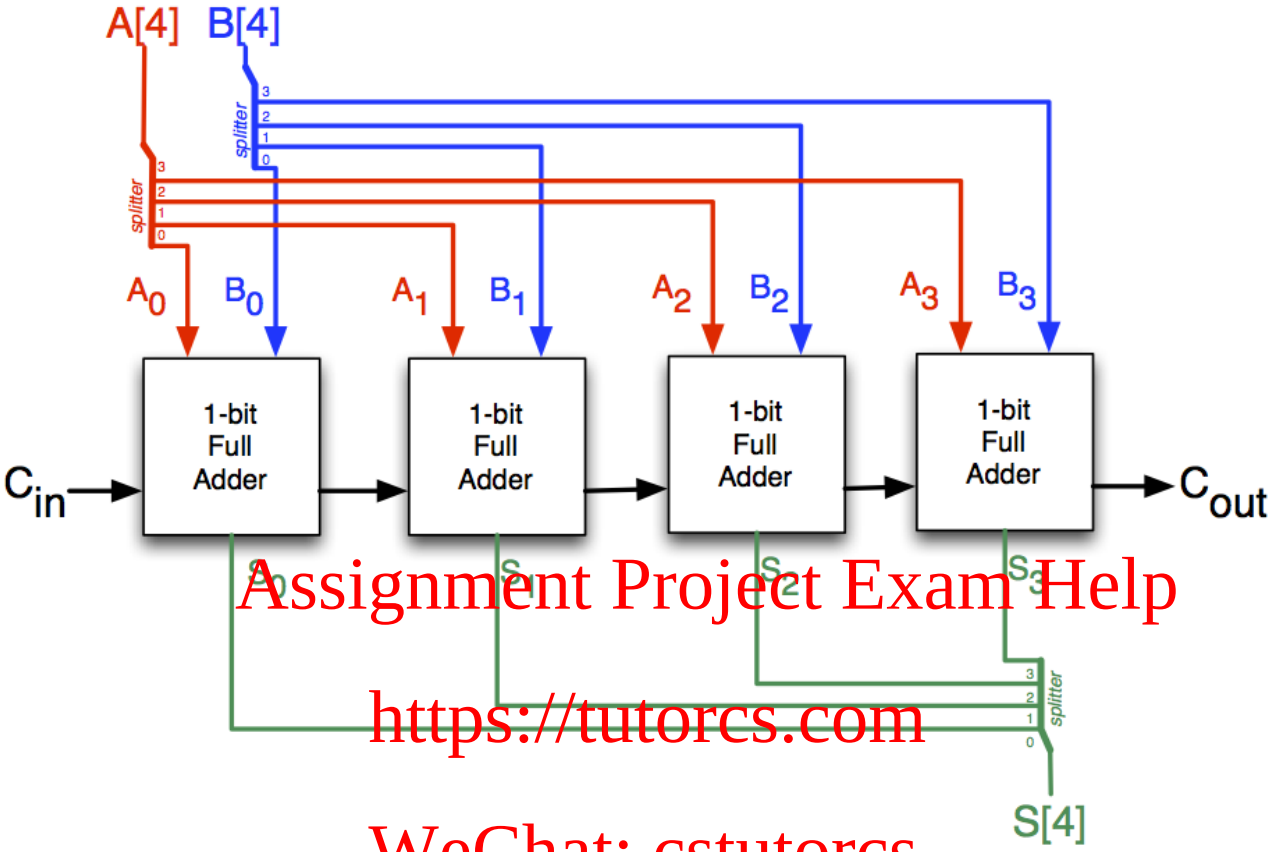


(https://canvas.cornell.edu/courses/42905/modules/items/1511779)
(https://canvas.cornell.edu/courses/42905/modules/item

Checkoff #2: Show your 1-bit adder circuit to your TA

Build a 4-bit adder

A 4-bit adder is as simple as cascading 4 one-bit adders together, with carry-out from one adder fed into the carry-in of the other adder.



Create the 4-bit adder in Logisim, re-using the 1-bit adder as a sub-circuit. If you ever need help, feel free to ask a TA for help. Save your work in a file called **four_bit_adder.circ**.

Hint: If you go into the **Wiring** folder, you'll be able to find **Splitters** there, which will be needed for this step.

Checkoff #3: Show your 4-bit adder circuit to your TA

Woohoo, you're done!

Tips

1: More on splitters

Splitters create a correspondence between a multi-bit value and several separate subsets of those bits. Despite the name, you may use the splitter as a "bundler" as well, joining multiple individual bits into a multi-bit value. Refer to the help page for the **Splitter** (http://www.cburch.com/logisim/docs/2.6.0/en/libs/base/splitter.html) for more information.

2: Use probes

Probes are great for debugging circuits. You can connect them to the in/out buses and set them to display values in base 10 for quick checking. Find them in the “Wiring” folder, and refer to the help page for the **Probe** (http://www.cburch.com/logisim/docs/2.6.0/en/libs/base/probe.html) if you need more information regarding them.

Tunnels are not strictly necessary for this lab, but they become important when the size of your circuit gets large. A "tunnel" acts like a wire that binds points with the same label together, except that the wire is not explicitly displayed. This is helpful to keep your circuit clean and organized. For more information, refer to the help page for the **Tunnel (http://www.cburch.com/logisim/docs/2.6.0/en/libs/base/tunnel.html)** . Feel free to practice creating one here.

Warning: Tunnels should always be used sparingly and thoughtfully; excessive use of tunnels will lead to docked points on future assignments.

4: Use the **Logisim Design Guidelines (https://canvas.cornell.edu/courses/42905/pages/logisim-design-guidelines)**

Remember, style matters! These links will help you get full style points on your projects AND learn how to get the most out of Logisim.

[next] **Lab 2: Left Shift 32 (https://canvas.cornell.edu/courses/42905/assignments/381696)** **(https://canvas.cornell.edu/courses/42905/assignments/381697)**