



Lab 5

[Computer Architecture I](#) [ShanghaiTech University](#)

[Lab 4](#) Lab 5 [Lab 6](#)

Objectives

- Practice designing and debugging basic digital logic circuits in Logisim
- Gain more experience designing and debugging circuits with both combinational logic and stateful elements
- Gain experience designing FSMs and implementing them as digital logic

Setup

Pull the starter files from [GitLab](#)

```
git clone https://autolab.sist.shanghaitech.edu.cn/gitlab/cs110/lab-5-starter.git
```

Some files are managed by [Git Large File Storage](#), you need to install it to pull all the files.

IMPORTANT: Please use the .jar file we've given you! And a note: Logisim does not save your work as you go along, and it does not automatically create a new .circ file when you open it! Save when you start, and save frequently as you work.

You can open Logisim via Java:

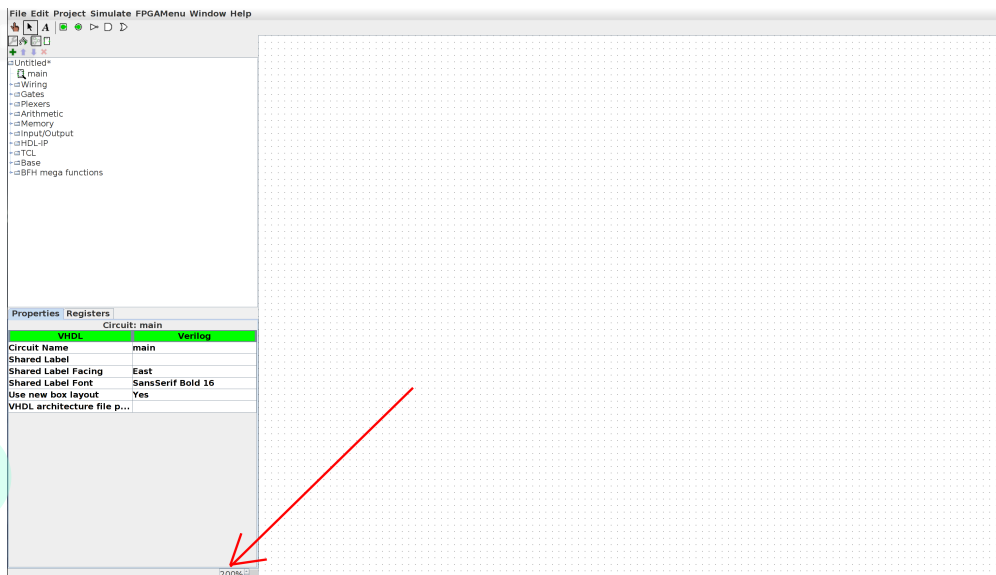
```
java -jar logisim-evolution.jar
```




Note: Logisim Evolution checks for internet connectivity by opening a HTTP connection to google before update. Since it attempts to access Google, it has to wait for some time until it finally knows we are not connected to Google. To disable this check, append `-nouupdates` (i.e. `java -jar logisim-evolution.jar -nouupdates`) to your start up command.

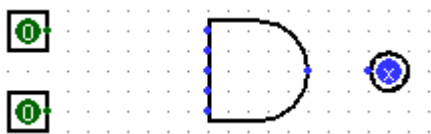
Exercises


Part 0: Warm Up

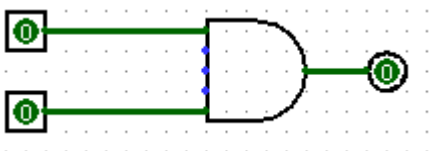
We'll begin by creating a very simple circuit just to get the feel for placing gates and wires. Before you start, take note of a useful feature: the zoom function! It's in the bottom left corner, and will make your life much easier for the next couple weeks.




1.  Start by clicking the **AND** gate button. This will cause the shadow of an **AND** gate to follow your cursor around. Click once within the main schematic window to place an **AND** gate.
2.  Click the **Input Pin** button. Now, place two input pins somewhere to the left of your **AND** gate.
3.  Click the **Output Pin** button. Then place an output pin somewhere to the right of your **AND** gate. Your schematic should look something like this at this point:



4.  Click the **Select tool** button. Click and drag to connect the input pins to the left side of the **AND** gate. This will take several steps, as you can only draw vertical and horizontal wires. Just draw a wire horizontally, release the mouse button, then click and drag starting from the end of the wire to continue vertically. You can attach the wire to any pin on the **AND** gate on the left side. Repeat the same procedure to connect the output of the **AND** Gate (right side) to the output pin. After completing these steps your schematic should look roughly like this:



5.  Finally, click the **Poke** tool and try clicking on the input pins in your schematic. Observe what happens. Does this match with what you think an **AND** Gate should do? Note that poking the wires themselves tells you the current value on that wire; this will be very useful later when you build more complex circuits.

Part 1: Sub-Circuits

Just as C programs can contain helper functions, a schematic can contain subcircuits. In this part of the lab, we will create several subcircuits to demonstrate their use.

IMPORTANT NOTE: Logisim Evolution guidelines say you cannot name a subcircuit after a keyword (e.g. `NAND`), also circuit names must start with "A-Za-z", so no numbers.

Action Item

Follow the steps below and show your final circuit to your TA at checkoff (remember to save!):

1. Open up the Exercise 1 schematic (`File->Open->ex1.circ`).
2. Open up the NAND1 empty subcircuit by double clicking on the name `NAND1` in the circuit selector in the left menu. (note the `1` at the end; because there is a component called `NAND`, we cannot call it `NAND`).
3. In the new schematic window that you see create a simple `NAND` circuit with the 2 input pins on the left side and the output pin on the right side. Do this without using the built-in `NAND` gate from the Gates folder (i.e. only use the `AND`, `OR`, and `NOT` gates provided next to the selection tool icon). You can change the labels for the inputs and output by selecting the input/output using the select tool and changing the property `Label` in the bottom left of the window.
4. Repeat these steps to create several more subcircuits: `NOR`, `XOR`, `2-to-1 MUX`, and `4-to-1 MUX` in the given skeletons. Please do not change the names of the subcircuits or create new ones; do you work in the respectively named circuit or else the autograder will not work properly. Do not use any built in gates other than `AND`, `OR`, and `NOT`. Once you've built a subcircuit, you may (and are encouraged to) use it to build others. You can do this by clicking and placing your created subcircuit like you would any other component. Note: for the `4-to-1 MUX`, `Se11` and `Se12` correspond to the 1st and 2nd bits of the 2-bit selector, respectively.

Hint: Try writing a truth table. You might also find the lecture slides useful for a refresher on how to build these. *You may want to consider using some of your custom subcircuits when designing the others.*

Checkoff

- Show your five circuits (`NAND`, `NOR`, `XOR`, `2-to-1 MUX`, and `4-to-1 MUX`) to your TA.

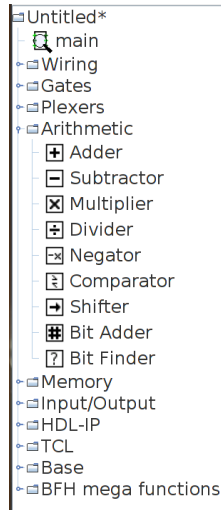
Part 2: Storing State

Let's implement a circuit that increments a value ad infinitum. The difference between this circuit and the circuits you've built for lab so far is that it will **store** this value in the **state** of a **register**.

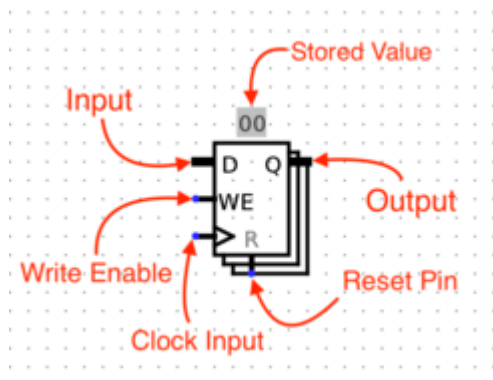
Action Item

The following steps will show you how to add registers to your circuit. Complete the steps and show the final circuit to your TA (remember to save!):

1. Open up the Exercise 2 schematic (File->Open->ex2.circ) and go to the empty AddMachine circuit.
2. Load in the Arithmetic Library if it is not already loaded (go to Project->Load Library->Built in Library and select Arithmetic). This library contains elements that will perform basic mathematical operations. When you load the library, the circuit browser at left will have a new Arithmetic folder.



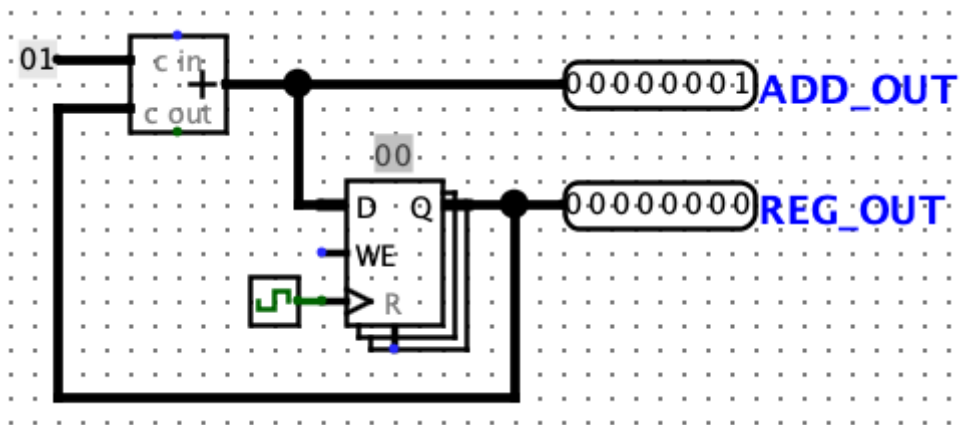
3. Select the adder subcircuit from the Arithmetic library and place the adder into your AddMachine subcircuit.
4. Load in the Memory Library if it is not already loaded (go to Project->Load Library->Built in Library and select Memory). This library contains memory elements used to keep state in a circuit. A new Memory folder will appear in the circuit browser.
5. Select the register from the Memory folder and place one register into your subcircuit. Below is an image diagramming the parts of a register.



6. Connect a clock to your register. You can find the clock circuit element in the Wiring folder in the circuit browser.
7. Connect the output of the adder to the input of the register, and the output of the register to the input of the adder.
 - o You may get a "Incompatible widths" error when you try to connect components. This means that your wire is trying to connect two pins together

with different bit widths. If you click on the adder with the [Selection](#) tool, you will notice that there is a [Data Bit Width](#) property in the bottom left field of the window. This value determines the number of bits each input and output the adder has. Change this field to 8 and the "Incompatible widths" error should be resolved.

8. Wire an 8-bit constant 1 to the second input of the adder. You can find the [Constant](#) circuit element in the [Wiring](#) library.
9. Connect the two output pins to your circuit so that you may monitor what comes out of the adder and the register. The output of the adder should be connected to [ADD_OUT](#) and the output of the register to [REG_OUT](#). Thus, by the end, your circuit should look like as follows:



10. Now start running your circuit by going to [Simulate->Ticks Enabled](#) (or [Command/Control + K](#)). Your circuit should now be outputting a counter in binary form.
11. If you want to run your circuit faster, you can change the tick frequency in [Simulate->Tick Frequency](#).

Checkoff

- Show your [AddMachine](#) circuit to your TA.

Part 3: FSMs to Digital Logic

Now we're ready to do something really cool: translate a FSM into a digital logic circuit.

For those of you who need a reminder, FSM stands for Finite State Machine. FSM's keep track of inputs given, moves between states based on these inputs, and outputs something everytime something is input.

We use a register to store the state of the FSM we're currently in, and combinational logic to map FSM input & current register state to FSM output & next register state.

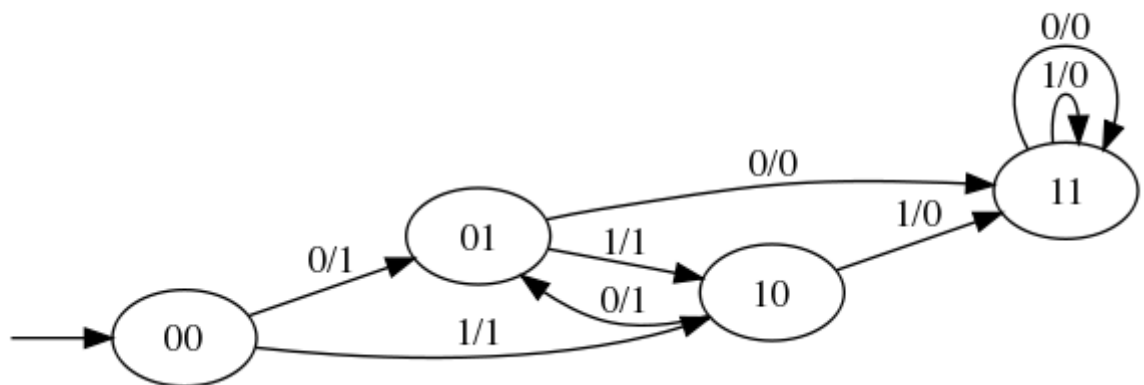
Action Item

Load the given starter file `ex3.circ` into Logism. Modify this circuit's subcircuits `StateBitZero` and `StateBitOne` to implement the following FSM:

If two ones in a row or two zeroes in a row have ever been seen, output zeros forever. Otherwise, output a one.

Show this completed circuit to your TA (remember to save!)

1. Note that the FSM is implemented by the following diagram (the four state names `00`, `01`, `10`, `11` are just names written in binary - they have no direct relationship with the actual zeros and ones of the FSM input/output). Take some time to understand how this diagram implements the FSM:



2. Observe that the following is a truth table for the FSM (convince yourself of this):

st1 st0 input next st1 next st0 output

0	0	0	0	1	1
0	0	1	1	0	1
0	1	0	1	1	0
0	1	1	1	0	1
1	0	0	0	1	1
1	0	1	1	1	0
1	1	0	1	1	0
1	1	1	1	1	0

3. We've provided you with a starter Logisim circuit to start out in `ex3.circ`.
4. Note that the top level of the circuit looks almost exactly the same as our previous adder circuit, but now there's a `FSMLogic` block instead of an adder block. `FSMLogic` is the combinational logic block for this FSM. We have handled the output bit for you, as it's the most complicated to simplify and implement. You should complete the circuit by completing the `StateBitOne` and `StateBitZero` subcircuits, which produces the next state bits.

Checkoff

- Show your `StateBitZero` circuit to your TA and demonstrate that it behaves correctly.
- Show your `StateBitOne` circuit to your TA and demonstrate that it behaves correctly.

Testing

To test, run the testing script via:

```
./test.sh
```

Since Logisim will be running in one terminal window already, make sure to open up a new window to run the testing script. If it says you don't have permission, run:

```
chmod +x test.sh
```

This script will copy your circuits into a testing harness, run your circuits on different inputs, and compare your results to ours. Therefore, please do not touch anything in the `testing` folder unless a TA instructs you to do so. However, you are more than welcome to check out the circuitry involved in testing your code as you will encounter it again when working on Project 2.

Schwertfeger, Sören <`soerensch AT shanghaitech.edu.cn`>

Chundong Wang <`wangchd AT shanghaitech.edu.cn`>

Modeled after UC Berkeley's CS61C.

Last modified: 2020-04-07