Lab 6

Deadline: EOD Tuesday, March 12th

Objectives

- TSWBAT practice designing and debugging basic digital logic circuits in Logisim
- TSW gain more experience designing and debugging circuits with both combinational logic and stateful elements
- TSW gain experience designing FSMs and implementing them as digital logic

Setup

Pull the Lab 10 files from the lab starter repository with

```
git pull staff master
```

All the work in this lab will be done from the digital logic simulation program **Logisim Evolution**, which is included in the lab starter files.

IMPORTANT: Please use the .jar file we've given you, not the version of Logisim that is downloaded on the lab computers! 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 Logism via:

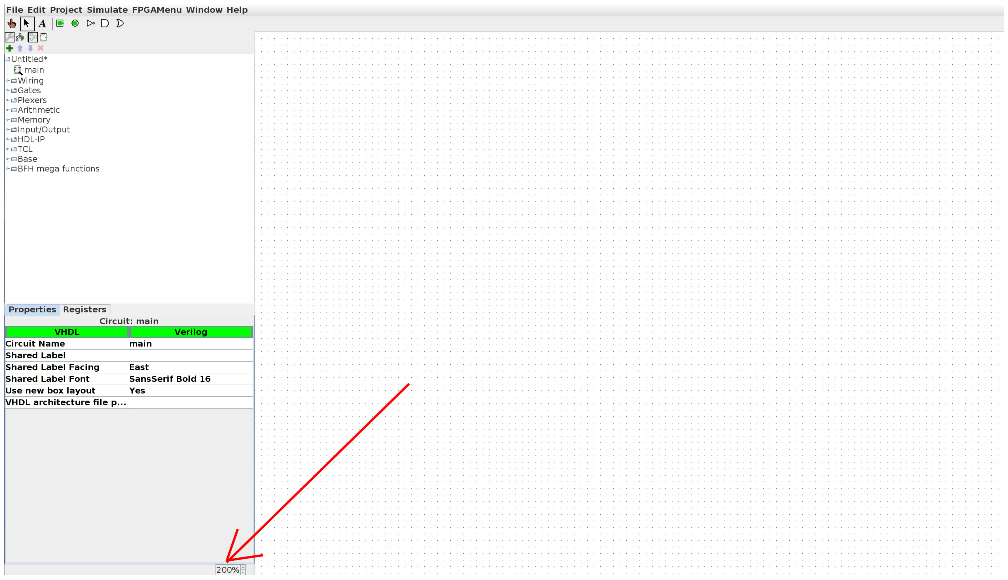
```
java -jar logisim-evolution.jar
```

IMPORTANT: Logism is a Java program that requires a GUI, so doing the lab over terminal won't work (without window forwarding, detailed below). If you wish to work on the lab locally, ensure you have Java installed on your local machine, and pull the latest lab starter files to your local machine. Then, you should be open the program as above. If you wish to run the program over the terminal, you will need to add the `-X` flag to your `SSH` command to enable window forwarding (for example, `ssh -X cs61c-xxx@...`). On Windows machines, you may need to additionally install Xming.

Introduction

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.



Circuit: main

VHDL	Verilog
Circuit Name	main
Shared Label	
Shared Label Facing	East
Shared Label Font	SansSerif Bold 16
Use new box layout	Yes
VHDL architecture file p...	

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 down 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?

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 guidlines 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. Create a new schematic (`File->New`) for your work.

2. Create a new subcircuit (`Project->Add Circuit`). You will be prompted for a name for the subcircuit; call it `NAND1` (note the `1` at the end; because there is a component called `NAND`, you cannot call it `NAND`).

3. In the new schematic window that you see create a simple `NAND` circuit with 2 input pins on the left side and an 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. Go back to your `main` schematic by double-clicking `main` in the circuit selector at the left of the screen. Your original (blank) schematic will now be displayed, but your `NAND1` circuit has been stored.

5. Now, single click the word `NAND1` in the list. This will tell Logisim that you wish to add your `NAND1` circuit into your `main` circuit.

6. Try placing your `NAND1` circuit into the `main` schematic. If you did it correctly, you should see a gate with 2 input pins on the left and one output pin on the right. Try hooking input pins and output pins up to these and see if it works as you expect.

7. Repeat these steps to create several more subcircuits: `NOR`, `XOR`, `2-to-1 MUX`, and `4-to-1 MUX`. 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. 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.*

Checkpoint

At this point, make sure that you are comfortable with the Logisim environment, creating sub-circuits, and re-using such circuits in other circuits.

Part 2: Storing State

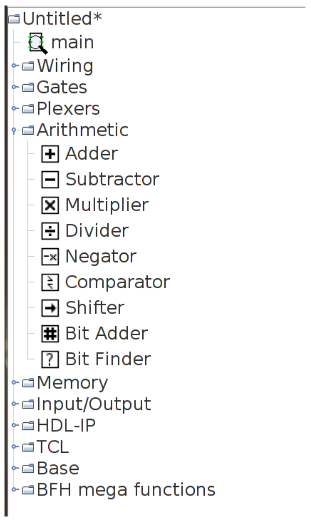
Let's implement a circuit that increments a value ad infinitum. The difference between this circuit and the circuits you've built for lab so far is that it will **store** this value in the **state** of a **register**.

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. Create a new subcircuit (`Project->Add Circuit`). Name this new subcircuit, `AddMachine`.

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 diagraming 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.
   - 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. Add two output pins to your circuit so that you may monitor what comes out of the adder and the register. Make sure the output is 8 bits. Thus, by the end, your circuit should look like as follows:

10. Now let’s see if you built your circuit correctly. Go back to the `main` subcircuit by double clicking on `main` in the circuit browser.

11. Single click on your `AddMachine` circuit to select it.

12. Change the `Facing` property to another direction. Any circuit with the `Facing` property can be rotated to accomodate wires as you need them. This will definitely be useful when you do your project.

13. Place your `AddMachine` subcircuit into the `main` subcircuit.

14. Select the `AddMachine` subcircuit you just placed into `main`.

15. Connect output pins to the `AddMachine` subcircuit. Output pins are ordered top to bottom, left to right. Thus, if you followed the schematic above, then the top pin on the right side outputs the value of the adder, and the bottom pin is the output of the register.

16. Right click on your `AddMachine` subcircuit, and select `View AddMachine`. This is the **ONLY** method to preserve state (i.e. keep register values at its current value). Double-clicking on the circuit at the circuit browser at left makes Logisim think you want to edit the circuit instead of just checking what state the circuit has.

17. Initialize the register value to `1`. You can do this by first, clicking on the register value with the poke tool. Then, type the hex value in.

18. To return to the main circuit while preserving state, go to `Simulate->Go Out To State->main`. Alternatively, you can use the keyboard shortcut `Command/Control + Up Arrow`

19. 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.

20. If you want to run your circuit faster, you can change the tick frequency in `Simulate->Tick Frequency`.

Checkpoint

At this point, make sure that you are comfortable with designing and simulating simple digital logic circuits in Logisim environment that use a mix of *combinational logic* and *state elements* (registers).

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 `FSM.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 `FSM.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.

Checkpoint

At this point, you should have more familiarity with designing and implementing FSMs, and the close relationship between FSMs and digital logic.

Advanced Logisim

Setup

Feel free to do each part as separate sub-circuits in the same Logisim file.

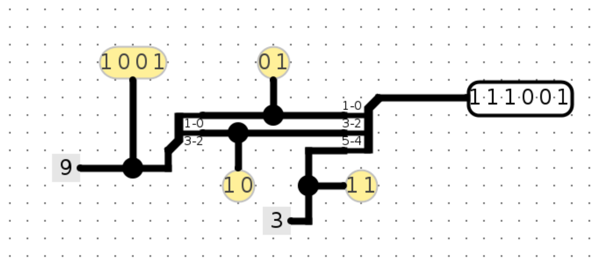
The following parts will introduce you to more advanced techniques/concepts in Logisim.

Advanced Features

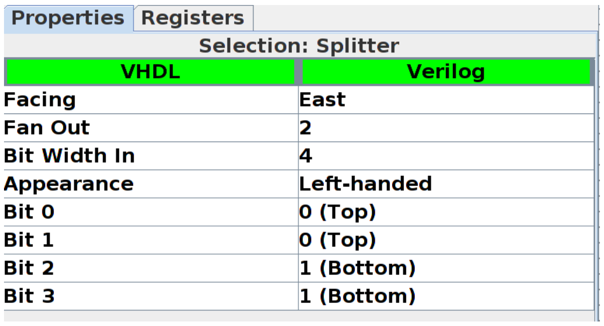
Here are three Logisim features that should both save you a lot of time and make your circuits look much cleaner.

Splitters

Splitters allow you to take a multi-bit value and split it up into smaller parts, or (despite the name) combine multiple values that are one or more bits into a single value. Here, we split the 4-bit binary number `1001` into `10` and `01`, then recombine it with `11` into the final 6-bit number `111001`:



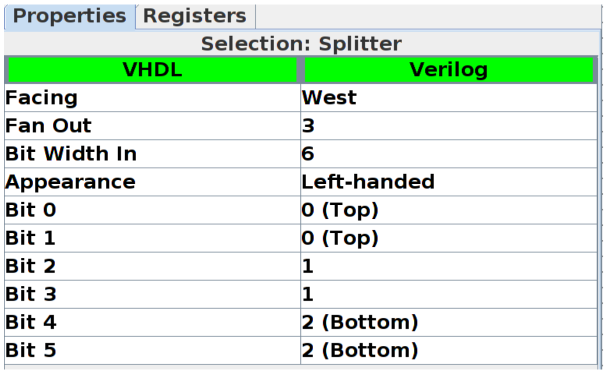
Click on a splitter to get its menu in the sidebar. You can use this menu to determine the number of arms on your splitter and how many bits should go on each arm. For the circuit above, the left splitter's menu looks like this:



Selection: Splitter

VHDL	Verilog
Facing	East
Fan Out	2
Bit Width In	4
Appearance	Left-handed
Bit 0	0 (Top)
Bit 1	0 (Top)
Bit 2	1 (Bottom)
Bit 3	1 (Bottom)

While the right splitter's menu looks like this:



Selection: Splitter

VHDL	Verilog
Facing	West
Fan Out	3
Bit Width In	6
Appearance	Left-handed
Bit 0	0 (Top)
Bit 1	0 (Top)
Bit 2	1
Bit 3	1
Bit 4	2 (Bottom)
Bit 5	2 (Bottom)

Notice that there's an option called `facing`. You can use this to rotate your splitter. Above, see that the splitter on the right is facing West while the splitter on the left is facing East.

If you see an error wire that is orange, this means that your bit width in does not match your bit width out. Make sure that if you're connecting two components with a wire, you correctly set the bit width in that component's menu.

Tunnels

A tunnel allows you draw an "invisible wire" to bind two points together. Tunnels are grouped by case-sensitive labels give to a wire. They are used to connect wires like so:

Which has an effect such as the following:

Some care should be taken as to which wires are connected with tunnels to which other wires, such as in this case:

Which in turn has the following effect:

We *strongly* recommend you use tunnels with Logisim, because they make your circuits much cleaner looking, and therefore easier to debug.

Extenders

When changing the width of a wire, you should use a bit extender for clarity. For example, consider the following implementation of extending an 8-bit wire into a 16-bit wire:

Whereas the following is much simpler, easier to read, and less error-prone:

Additionally consider the case of throwing out bits. In this example, an 8-bit wire is being converted into a 4-bit wire by throwing out the other bits:

Despite the implications of its name, a bit extender can also do this same operation:

Part 4: Practice with Splitters

We're going to construct a circuit that manipulates an 8-bit number.

Action Item

Complete the following steps to create the splitter circuit, and show this to your TA (remember to save). When you've completed the circuit, answer the question in step 11.

1. Create a new subcircuit and name it `Ex4`.

2. Add an 8-bit input pin to your circuit and label it `In1`.

3. Add a 1-bit output pin labeled `Out1` and an 8-bit output pin labeled `Out2` to your circuit.

4. Go to the `Wiring` folder and select the `Splitter` circuit. This circuit will take a wire and split it into a set of wires of smaller width. Conversely, it can also take many sets of wires and combine them into a larger bus.

5. Before you place your circuit, change the `Bit Width In` property (bus width) to `8`, and `Fan Out` property (# of branches) to `3`. If you move your cursor over the schematic, your cursor should look as follows:

6. Now, select which bits to send out to which part of your fan. The least significant bit is bit `0` and the most significant bit is bit `7`. Bit `0` should come out on fan arm `0`, bits `1`, `2`, `3`, `4`, `5` and `6` should come out on fan arm `1`, and bit `7` should come out on fan arm `2`. FYI: the `None` option means that the selected bit will not come out on ANY of the fan arms.

7. Once you configure your splitter, you can place your splitter into your circuit.

8. Route `In1` to the splitter. Attach a 2-input `AND` gate to fan arms `0` and `2` and route the output of the `AND` gate to `Out1`.

9. Now, interpret the input as a “sign and magnitude” number. Place logic gates and other circuits to make `Out2` to be the negative “sign and magnitude” value of the input. Sign and magnitude is an alternate way of representing signed values - like 2’s Complement, but simpler! The combinational logic should be straight-forward.

10. We will need another splitter to recombine the fans into a single 8-bit bus. Place another splitter with the proper properties (Bit Width In: 8, Fan Out: 3, correct fan widths). Play with the `Facing` and `Appearance` properties to make your final circuit as clean-looking as possible.

11. Answer the following question:

    If we decide to take the input and interpret it as a 2’s Complement number, what inputs will produce `Out1 = 1`? *Hint: What do the first and last bits of a 2’s Complement number being 1 tell you about the number?*

Part 5: Rotate Right

With your knowledge of splitters and your knowledge and experience with multiplexers from way back in Lab 3, you are ready to implement a non-trivial combinational logic block: `rotr`, which stands for “Rotate Right”. The idea is that `rotr A,B` will “rotate” the bit pattern of input `A` to the right by `B` bits. So, if `A` were `0b1011010101110011` and `B` were `0b0101` (`5` in decimal), the output of the block would be `0b1001110110101011`. Notice that the rightmost `5` bits were rotated off the right end of the value and back onto the left end. In RTL, the operation would be something like `R = A >> B | A << (16 - B)`.

Action Item

Implement a subcircuit named `rotr` with the following inputs:

- `A` (16-bit), the 16-bit input to be rotated
- `B` (4-bit), the rotation amount (why 4 bits?) Show off your `rotr` subcircuit in the `main` subcircuit.

The output should be `A` rotated right by `B` bit positions, as outlined above. You are **NOT** allowed to use Logisim shifters in your solution, though all other combinational logic (MUXes, constants, gates, adders, etc.) is allowed. Logisim’s built-in MUXes (find them under the `Plexers` menu) might be especialy helpful. Your solution shouldn’t involve a clock or any clocked elements, like registers.

Hint 1: Before you start wiring, you should think very carefully about how you might decompose this problem into smaller ones and join them together. You should feel very free to use subcircuits when implementing `rotr`. If you don’t, expect to regret it.

Hint 2: Just because we gave you an RTL representation doesn’t mean it’s the best way to look at this problem. Think about the input bits of `B` and think about how to effectively use splitters! Can you do something with the binary form? Remember why binary is good for use in computers: a `1` is easy to represent as an `ON` signal, and a `0` is easy to represent as an `OFF` signal. Let’s say we want to rotate `9` times. `9` is `1001` in binary, or `1*8 + 0*4 + 0*2 + 1*1`. Can you use this to make a cleaner circuit?

Checkoff

Show your working final circuits for the five parts of this lab to your TA.