CS447 Lab #10 (Week 11)

Introduction

For this lab, you will implement a finite state machine (FSM) in Logisim.

Note: This Lab must be submitted online by 11:59 pm on Thursday, November 17th. Please see the instructions on the final page.

Objective: Implement a finite state machine in Logisim

Features of FSMs

Finite state machines (FSMs) were covered in the lecture slides for 11/3. Unlike simple logic gates, which implement simple mappings between inputs and outputs, simple FSMs (that is, Moore machines[1]) use inputs to determine the transitions between system states, and the states determine the outputs.

An FSM should include the following features:

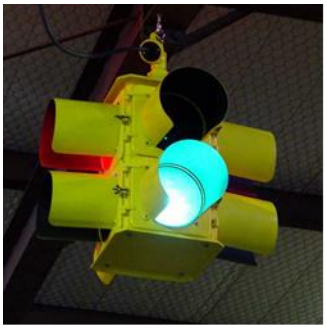
1. An **element to hold the state** information (for example, a register).
    - An ***n*-bit storage element** can represent up to **2^n possible states**.
2. A **state transition** table: Mapping of [inputs, state] → [new state]
    - The text also uses the term "next state function"
3. A **clock** to determine when the state storage element is updated to the next state.
4. An **output table** to determine what output the FSM should produce, given a particular state.

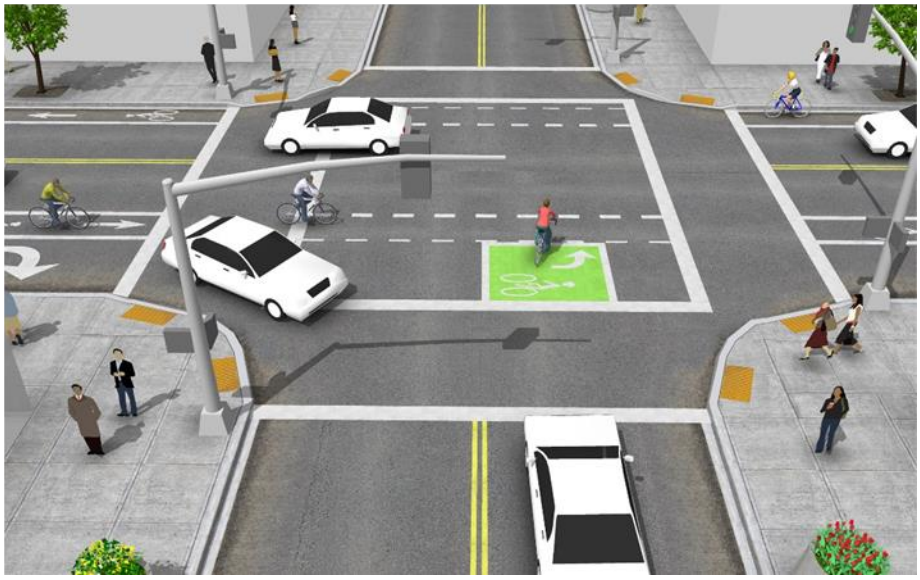
Example FSM

For this lab, we will use the traffic light example from the textbook. This example assumes a two-way light with only red and green signals, which sits at the intersection of a North → South road and an East → West road[2]:

[1] For this lab, we will assume that the FSM is a Moore machine, so that the outputs only depend on the states. This contrasts to Mealy machines, for which the outputs rely on both the inputs and the state.

[2] Image credits: http://www.nacto.org, http://www.kbrhorse.net/signals/terms02.html





Assume that there is a single controller for all the lights: the "NS" lights that face the north and the south, and the "EW" lights that face the east and the west. This controller can adopt one of two possible **state** values, which might be conceptualized as a command that the controller is internally maintaining: Set the NS lights to green (and the EW lights to red), or set the EW lights to green (and the NS lights to red). We can describe this state with a single bit (0 or 1). Each possible value of this bit can be assigned one of the two state labels:

0. **NSgreen**: NS lights = green, EW lights = red.
1. **EWgreen**: EW lights = green, NS lights = red.

Since we are assuming that this is a Moore machine, the state of the controller directly maps onto the controller's output. This can be conceptualized as the translation of the command into the actual implementation. We will use one signal to switch the NS lights to green or red, and one signal to switch the EW lights to green or red:

- **NSlite**: 1: NS lights = green; 0: EW lights = red
- **EWlite**: 1: EW lights = green; 0: NS lights = red

This mapping between states and outputs is represented in the Output Table:

	Outputs	
	NSlite	EWlite
NSgreen	1	0
EWgreen	0	1

Continuing from the textbook example, we will assume that the controllers receive **inputs** from detectors placed underneath the roads. The detectors detect the presence of cars. Again, we can represent these as single bits.

1. **NScar**: If this bit = 1, then a car has been sensed by the detector under the north-south road.
2. **EWcar**: If this bit = 1, then a car has been sensed by the detector under the east-west road.

Only the NW lights or the EW lights should be set to green at any one time. The other light should be set to red. When a car arrives on a particular road, the light for that road should be set to green. It should not change to red – and the opposite light should not change to green – until a car is detected on the other road.

This description of the desired behavior gives us sufficient information to build the **state transition** table, which maps the [state, input] information to the appropriate new state:

	Inputs		
	NScar	EWcar	Next state
NSgreen	0	0	NSgreen
NSgreen	0	1	EWgreen
NSgreen	1	0	NSgreen
NSgreen	1	1	EWgreen
EWgreen	0	0	EWgreen
EWgreen	0	1	EWgreen
EWgreen	1	0	NSgreen
EWgreen	1	1	NSgreen

This transition table specifies how the controller should behave in all situations. For example, assume that we are in NSgreen with no detected cars, and then cars are detected by both the NScar and EWcar detectors. In this case, the transition table tells us that the state should be changed to EW green. Also note that a graphical representation of this FSM is available on page B-70 of the textbook.

Implementing the traffic light FSM

Step #1: Create an element to store the controller's state, and some input and output pins.

In Logisim, we will need to create different circuits, on different canvases. As a guideline, refer to the "door FSM.circ" file for 11/3 (see materials zip file on the course website). Following from that example, create canvases for the FSM, Output, State Transition, and Main circuits.

In the **FSM canvas**, create a memory element to hold the 1-bit state. This can be a register or a D flip-flop. At this point you can also create and connect an input pin for the Clock input (which will be sent in from Main).

In the FSM circuit of Door FSM, note that there are inputs for the detectors that the Door example uses, and output pins for output units that the door needs to control. Our traffic light example will take inputs from two road detectors, and will need to control the output of two sets of lights. As in Door FSM, we'll also create an output to show the current state of the memory element, so that we can visualize this state from within the main circuit.

For all of these inputs and outputs, you will need to create input/output pins in your FSM circuit. Don't worry about adding any wires yet.

Step #2: Create a circuit to implement the Output Table.

For reference, the table from above is copied here.

	Outputs	
	NSlite	EWlite
NSgreen	1	0
EWgreen	0	1

This table represents a mapping from a 1-bit input to two separate two bit outputs. Therefore, in the **Output circuit**, you will need to create:

- One input pin, to represent the state (assume 0 = NSgreen and 1 = EWgreen).
- Two output pins, to represent the signals that switch the NS lights and the EW lights.
- Wires and a logic gate to implement the table logic.

You can test the Output circuit by confirming that it recreates all the mappings in the Table.

Step #3: Create the initial components of the main circuit

Again, the Door FSM example can serve as a partial guide in setting up the initial components of the main circuit. At this stage, you can add the following components:

- An instance of your FSM circuit
- Input pins that will allow you to manually poke the two road detector units
- The clock
- LEDs* that will signal the color of the NSlite and the EWlite

- Some means of displaying the current system state. You may use either an output pin or an LED (as in Door FSM) for this purpose.
- Labels, to clarify the roles of the different input/output pins.

*The LEDs may be found in the Input/Output menu. Set "On Color" to green and "Off Color" to red. Click on the fields for these Attributes to bring up a color wheel; Logisim will automatically populate the field with the hexdecimal representation for the color you choose.

For now, don't worry about wiring the components together.

Step #4: Create the state transition circuit.

Here's the State Transition Table from above:

	Inputs		
	NScar	EWcar	Next state
NSgreen	0	0	NSgreen
NSgreen	0	1	EWgreen
NSgreen	1	0	NSgreen
NSgreen	1	1	EWgreen
EWgreen	0	0	EWgreen
EWgreen	0	1	EWgreen
EWgreen	1	0	NSgreen
EWgreen	1	1	NSgreen

Remember that this table encodes the mappings between three inputs (a single bit input for the Current State (NSgreen/EWgreen), and two single-bit inputs for the detector signals) and an output (a single bit output for the next state (NSgreen/EWgreen)). Therefore, in your **State Transition** circuit, you will need to create three input pins, and one output pin.

As an optional exercise, you might want to try finding a Boolean expression for this table by using a Karnaugh map. For this lab, you can use the expression provided by the textbook:

$$\text{NextState} = (\overline{\text{CurrentState}} \cdot \text{EWcar}) + (\text{CurrentState} \cdot \overline{\text{NScar}})$$

NextState = (NOT(CurrentState) AND EWcar) OR (CurrentState AND NOT(NScar))

In the **State Transition** circuit, you should implement this Boolean expression. **NextState** will be the output, and all the variables on the right will serve as inputs. The operators can be implemented with the appropriate logic gates. You can confirm that your circuit is working by comparing your output with the values in the State Transition table.

Step #4: Finish the FSM circuit

Add instances of your **State Transition circuit** and your **Output circuit** to the **FSM circuit**. At this point, you can wire everything together. The door FSM example may provide some guidance for this. In general, make sure to:

- Send a clock input to your storage element
- Provide all the inputs that your State Transition circuit and Output circuit expect
- Use all the outputs that your State Transition circuit and Output circuit provide

Since the memory element stores state, the Next State will be its input, and its output can be used to signal the Current State.

Step #5: Finish the main circuit.

At this point, you can wire together the individual components of your Main circuit. Each of the existing input/output pins, the LEDs, and the clock will need to be wired to the appropriate points on the FSM.

To test the main circuit, you can again confirm that the relationships within the State Transition Table hold, and also confirm that the states are associated with the expected LED outputs.

Submission:

If you finish during recitation, please notify the TA, so that your credit for this assignment can be confirmed right away.

Regardless of whether you finish during recitation or not, you must submit your work online. Save the file as **YourPittUserName_lab10.circ** and submit the file via the appropriate link in Courseweb (see Course Documents/Week 10 Lab). The lab must be submitted by **Thursday, November 17th**, by **11:59 pm**.