COMSM1302 Lab Sheet 4

This lab sheet covers content from week 4 lectures, focusing on transistors and finite state machines. On successful completion of the lab, students should be able to:

1. Use transistors as building blocks to create logic gates.
2. Build clocked circuits that utilise sequential logic.
3. Design circuits that implement finite state machines.

You'll need the Logisim software to implement circuits, and may need to install Java first if running Logisim on your own computer. Details on how to download and run Logisim can be found on the unit page and if you need help with using Logisim, try referring to the Logisim user guide.

Transistors

Build the following logic gates on Logisim, using transistors:

- NOT A
- A NAND B
- A AND B
- A NOR B
- A OR B

You should use 1-bit input pins, 1-bit output pins, p-type (facing south) and n-type (facing north) transistors, a *power* component for the Vdd signal, and a *ground* component for the Vss signal. All of these components can both be found in the *wiring* library in Logisim.

Rise/fall detector

Design a **rise/fall detector** in Logisim, using the one-shot circuit provided in lectures. The detector should be a clocked circuit with a 1-bit input in and two 1-bit outputs $rise$ and $fall$, that will have the following behaviour on a rising clock edge:

- If in has transitioned from low to high since the last rising clock edge, then $rise$ should pulse high for one clock cycle (as with a one-shot)
- If in has transitioned from high to low since the last rising clock edge, then $fall$ should pulse high for one clock cycle (as with a one-shot)

You should first try to build this using the existing one-shot circuit design as a subcircuit. Then, try to build this circuit by adapting the one-shot Mealy machine design directly.

Program counter

Implement a **program counter** on Logisim. It has three 1-bit inputs $reset$, inc and $load$, a clock input, a 16-bit input in, and a 16-bit output out. On a rising edge from the clock, the program counter behaves as follows:

- If $reset$ is high, then out is set to 0.
- If $load$ is high, then out is set to in.
- If inc is high, then out is set to $out + 1$.

At most one of $reset$, inc and $load$ will ever be high at once - you don't have to worry about what happens if more than one of them is high. After the rising edge, out remains constant (regardless of input behaviour) until the next rising edge.

Traffic lights

When a pedestrian presses the button on a traffic light at most pedestrian crossings, the lights will cycle through different states. These states are $green \Rightarrow amber \Rightarrow red \Rightarrow flashing amber \Rightarrow green$, with a set amount of time between each transition. A good way of encoding these four states is with a *one-hot* (not "one-shot") encoding. This type of encoding uses one bit per possible state, and sets exactly one bit high at any given time:

state	encoding
green	1000
amber	0100
red	0010
flashing amber	0001

Your goal is to build a circuit that will conform to this specification by following the steps detailed below:

1. Draw a **state transition diagram**, where the transition between these states is determined by a 1-bit input *change*:

   - If *change* is high, the state should transition to the next state in the sequence
   - If *change* is low, the state should not transition to the next state in the sequence

   Is this a Moore or Mealy machine design?

2. Create a **circuit to store the state**. Your circuit should have one 1-bit input *change* and four 1-bit outputs to represent the four state bits. On each clock cycle, the outputs should change according to your state transition diagram. At startup, or at least after the first few clock cycles, the state should be green.

   Hint: You may find a one-shot useful to initiate the first state.

3. Create a **circuit to time the transitions**. Your circuit should have a 1-bit input *start*, a 16-bit input *time*, and a 1-bit output *done*. When *start* goes from 0 to 1, the circuit should interpret *time* as a binary number, wait that many clock cycles, then make *done* high until *start* goes from 1 to 0 again. You may assume *start* doesn't go from 1 to 0 until after *done* goes high.

   Hint: You may want to create a subcircuit to check whether two binary numbers are equal.

4. Create a **timed traffic light circuit**, using your designs from part 2 and 3 as subcircuits. You circuit should have a 1-bit input that initiates the following sequence:

   - The state should immediately transition from green to amber.
   - After approximately a 2 second delay, the state should then transition from amber to red.
   - After approximately a 15 second delay, the state should then transition from red to flashing amber.
   - After approximately a 5 second delay, the state should then transition from flashing amber to green.

   Your circuit should use clock speed of 512Hz and, while flashing, the amber light should alternate between 1 and 0 at a frequency of approximately 1.25Hz, starting with a lit light.

   Hint: You may want to create a subcircuit to generate the flashing behaviour.