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 a transistor based design on Logisim for each of the following logic gates:

- 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 (this should be based around 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 exhibit 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 circuit using the existing one-shot circuit design as a subcircuit (you may need more than one). Then, try to build this circuit by adapting the one-shot Mealy machine design directly.

Program counter

Implement a program counter on Logisim (this circuit will later form an important part of the Hack CPU). It has three 1-bit inputs *reset*, *inc* and *load*, a clock input, a 16-bit input *in*, and a 16-bit output *out*. 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. 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.

After the rising edge, *out* remains constant (regardless of input behaviour) until the next rising edge. You may use the pre-built versions of any circuit you have designed so far.

Traffic lights

When a pedestrian presses the button on a British traffic light at most pedestrian crossings, the lights will cycle through different states (colours). 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/colours is driven by a 1-bit input. If the input is high, the state should transition on the next clock cycle; if it is low, then no transition should occur. Is this a Moore or Mealy machine design?

2. Create a subcircuit $Brain$ to store the state. Your circuit should have one input, *change*, and four outputs for the four state bits. On each clock cycle, if *change* is high, then the output should change to the next state in the sequence (e.g. from flashing amber to green). At startup, or at least after the first few clock cycles, the state should be green. **Hint:** You may find a one-shot useful for this.

3. Create a clocked subcircuit $Timer$ with a 1-bit input *go*, a 16-bit input *time*, and a 1-bit output *done*. When *go* goes from 0 to 1, the circuit should interpret *time* as a binary number, wait that many clock cycles, then take *done* high until *go* goes from 1 to 0 again. You may assume *go* doesn't go from 1 to 0 until after *done* goes high. **Hint:** You may want to create another subcircuit which can check whether two binary numbers are equal.

4. Using your $Brain$ and $Timer$ subcircuits, create a traffic light circuit. You should take a single 1-bit input that signals an immediate transition from green to amber. When using a clock speed of 512Hz, the delay from amber to red should be approximately 2 seconds, the delay from red to flashing amber should be approximately 15 seconds, and the delay from flashing amber to green should be approximately 5 seconds. While flashing, the amber light should alternate between 1 and 0 at a frequency of approximately 1.25Hz, starting with a lit light. **Hint:** For testing at full clock speed, you may want to use an input pin with a one-shot attached.