COMSM1302 Lab Sheet 3

This lab sheet covers content from week 3 lectures, focusing on memory and storage. On successful completion of the lab, students should be able to:

1. Assemble standard memory components (R-S latch, D latch, D flip-flop, n-word RAM) on Logisim.
2. Adapt standard memory components so that they are active low/high or positive/negative edge-triggered.
3. Manipulate inputs of standard memory components so they store a desired value.

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.

You will also need to borrow a NAND board to try physical implementations, which you can find details of on the unit page. If you get stuck using NAND boards, try watching the NAND board explanation or have a look through the GitHub documentation.

Latches to flip-flops

Build each of the following circuits on Logisim using 2-input NAND gates and then transfer each design to a NAND board. You should test that you know how to update the saved value to 0 and to 1:

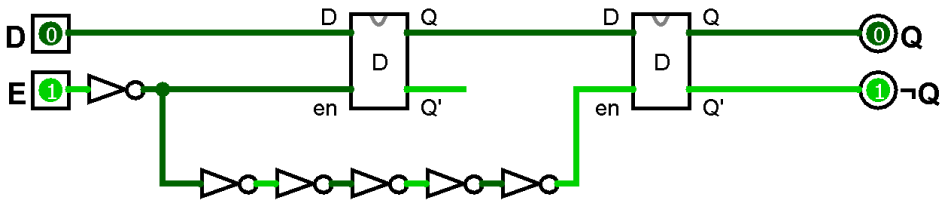
- Active low R-S latch
- Active high R-S latch
- Active high D latch
- Active low D latch
- Positive edge-triggered D flip-flop
- Negative edge-triggered D flip-flop

Hint: Half of these designs are explicitly covered in lectures!
Challenge: Can you connect your NAND board D-latch design with other students to implement a shift register? This will require a shared power source, using a USB splitter provided in the labs.

Treachery of Logisim

The following section should be carried out only on a NAND board, **not** on Logisim:

- Use your D flip-flop design from the last section (on a NAND board) and replace the NAND gate acting as a NOT gate between the two enable inputs with five NOT gates, as shown below:



- Now verify that the flip-flop behaves exactly as it did previously, as we would expect. After all, we all know that $\neg\neg x \equiv x$, right? ...right? *...what's gone wrong?*

- If you take the five chained "NOT gates" on your NAND board and link them in a ring (as shown below), can you explain the behaviour you observe?

This is a good illustration of how simulators like Logisim can break in situations where propagation delay becomes important — it won't model either of these situations accurately! For this reason, whenever you use a D flip-flop in Logisim as a building block in a larger circuit, you should use the **built-in** version (found in the "Memory" library) — it has special internal logic to keep it from getting too tangled up.

Memory

The Hack CPU has 32KB of memory divided into 16,384 words. That's a bit more than a single 1-bit register, but luckily it also makes a good example of the power of abstraction:

- Create an 8-word RAM circuit in Logisim using premade 16-bit registers (found in the "Memory" library). It should have a 16-bit input *in*, a 3-bit input *address*, a 1-bit input *load*, a clock input, and a 16-bit output *out*. On a rising clock edge, out should be set to the 16-bit value stored at address *address*; if *load* is high then this value should also be updated to *in*.
- Using your 8-word RAM circuit as a subcircuit, implement a 64-word RAM circuit in Logisim. This should have a similar configuration to your 8-word RAM, although *address* will now be a 6-bit input. **Hint:** You will need to split *address* into two parts!
- Using your 64-word RAM circuit as a subcircuit, create a 32KB RAM circuit in Logisim with 14 *address* bits. **Hint:** You may want to create more subcircuits to build up to 32KB RAM in steps (copy-and-paste is your friend)!

Other flip-flops

Besides D flip-flops, there are two other types of flip-flops in occasional use (that are non-examinable on this course) and we can create each from a single D flip-flop plus some extra logic! Build the following flip-flops, given the specifications below and starting from a pre-built D flip-flop for each:

- A *T flip-flop*'s only input is a clock signal. It has two outputs, Q and $Q' = \neg Q$, and Q is a clock signal with half the frequency of its clock input. Here T stands for "toggle".

  Once you have the circuit working, try using a "Clock" input (found in the "Wiring" library) for the input pin. To automatically oscillate the clock input, go to the "Simulate" menu and set "Tick frequency" to 2Hz and select "Ticks enabled". You can press Ctrl+T or click the clock input to manually trigger a clock edge.

- A *JK flip-flop* has three inputs: J, K and an *clock* (enable) input. On a rising edge from the clock, the outputs Q and Q' are set depending on J and K as follows:

J	K	Q	Q'
0	0	No change	No change
0	1	0	1
1	0	1	0
1	1	Toggle	Toggle

  Can you see how to implement this with just four AND/OR/NOT gates (as well as the D flip-flop)? **Hint:** You should allow the clock signal to pass through to the D flip-flop unchanged.