NAND 3/3: Storage

Kerstin Eder

September 27, 2019

This lab continues to use the NAND boards and Logisim. Last week, you made arithmetic logic, all of which were *combinatoric* (results appear as quickly as the gate delays allow). This week, you will make latches and flip-flops to store data. Flip-flops are *sequential*, meaning they are synchronous to a *clock signal.*

Goals of this lab

- Continue to familiarise yourself with Logisim for logic circuit design, simulation and testing.
- Transfer designs onto NAND boards, aiming for a one-to-one match of the implementation to the simulation. See the previous lab sheets for guidelines on the NAND board kits and the note below.
- Create synchronous logic that could be combined with combinatorial logic to create finite state machines.

Advice

Please note that, while the simulator offers a wide range of logic gates, the NAND boards only have NAND gates with *two* inputs. The fundamental idea is that the designs you build and test with the simulator are a one-to-one match with what you implement on the NAND boards. So, it is really important that your designs are directly implementable on the NAND boards, i.e. without modification of the logic. In particular, do not use 3-input NAND gates in your designs.

Also, please read through the entire lab sheet before starting. Important information is contained on the last page for you to successfully complete the task on **JK flip-flops**.

Storage

The following tasks require that you understand the material in the lecture on "Storage". Referring to the slides will assist you in completing them successfully, although you are also expected to do additional self-study in preparation of this lab.

The tasks move towards *storage* logic, starting with latches.

The SR latch has two inputs, S and R, and two outputs, Q and $\overline{Q}$. As per the transition tables (which you can find in the lecture slides), when $S = 0$ and $R = 0$, the current values on the outputs remain unchanged (HOLD). Asserting $S = 1$ will ***set*** $Q = 1$ and $\overline{Q}$ to its complement (SET). Asserting $R = 1$ will ***reset*** the latch so that $Q = 0$ (RESET). The condition $S = 1, R = 1$ is invalid and should be avoided.

1 Warm-up: SR latch

Implement an SR latch in both Logisim and on a NAND board. Take care with the fact that in the simplest NAND implementation, the inputs are treated as $\overline{S}$ and $\overline{R}$, so you may wish to invert the inputs in order for the circuit to be easier to understand.

Test your SR latch systematically, both in Logisim and on a NAND board.

2 Stepping stone: D-type latch

Now implement a D-type latch in Logisim and on a NAND board. A D-type latch has an input D, which is propagated to the output only when the enable input, E is set to logic 1. Changes to D when $E = 0$ should not change the outputs.

Building a D-type latch is a stepping stone to building a D-type master-slave flip-flop. It is strongly recommended that you perform this task, fully test your design and explain why it works to your lab partner before moving on.

3 D-type master-slave flip-flop

Using your D-type latch implementation, extend your design to create a D-type master-slave flip-flop. This will make your storage component *edge sensitive* to the E (or clock) input, rather than *level sensitive* and transparent. **Ensure you make a master-slave implementation, as there is more than one variety of D-type flip-flop.**

You should implement this, as usual, in both Logisim and on the NAND boards. When testing with the NAND boards notice how the *master* and *slave* parts of the circuit operate on different levels of the clock input.

Again, think carefully about how best to test that your solution works correctly.

Tips

When using Logisim, you may wish to use a `Clock` component from the `Wiring` library for your clock input, instead of a `Constant`. You can then make the clock "tick" by pressing `ctrl+T`, or have a free-running simulation by pressing `ctrl+K`. Of course, with the NAND boards, you will have to clock the device yourself by repeatedly connecting and disconnecting an input wire or toggling a switch.

4 Shift-register

This task requires combining several D-type master-slave flip-flops. You can do this in the lab with working D-type master-slave flip-flops of other groups. Before connecting designs, demonstrate to the other group(s) how your design works and show that it works correctly.

You can build a shift register by chaining together the Q output of your master-slave flip-flop to the D input of the next device, and so on. You will also have to chain the clock signals together.

A shift register *shifts* values along its chain. At time $T = 0$ cycles, if a value $D = 1$ is input into the left-most shift register, then at $T = 5$ cycles, that value will be present on the Q output of the *fifth* flip-flop in the chain. A history of the input sequence will therefore be visible across the shift register.

Again, think carefully about how best to demonstrate that your shift register works correctly.

5 JK flip-flop

This lab has, so far, focused on D-type devices, with a clock or enable input, and a data input. Your final task is to design, implement and build a device conforming to the *JK flip-flop* definition.

The JK flip-flop has *three* inputs: `J`, `K`, and `clock`. The characteristic table is as follows:

Make a version of this that is *positive edge triggered* and uses a master-slave configuration.

J	K	Description	Q_{next}
0	0	hold	Q
0	1	reset	0
1	0	set	1
1	1	toggle	$\overline{Q}$

Table 1: JK flip-flop characteristic table.

You will need to do your own research to establish how to make such a circuit. Conceptually, how does a JK flip-flop differ from other types of flip-flops?

Template

Logisim is a simple tool and has its own flip-flop libraries. However, this lab asks you to build your own from NAND gates. Logisim sometimes has difficulty resolving the state of wires if there are 'unknown' conditions several layers deep. To that end, simulating in Logisim is *slightly* tricky, but not impossible.

We provide a template, `jk-ff-template.circ` to get you started, and to overcome this problem. It is available from Blackboard.

Use the labelled inputs as they are intended. There are two feedback inputs that you will very likely need in your final design. Your first two NAND gates are provided — connect them to the rest of your design appropriately. To reset the design, simply set the Reset input to 1, then cycle the clock input (press ctrl+t twice). Then, return Reset to 0. This is known as a *synchronous reset*, because it is controlled by the clock.

Your NAND board implementation should work *without* needing the reset circuitry, therefore J, K and the feedback inputs can be *directly connected* to the first NAND gates.

6 Extension: A Ring Oscillator

If you find playing with logic is fun, good for you! As a reward for getting this far, implement a Ring Oscillator (see the lecture slides on **"Storage"**) using your NAND board(s). Here are some questions for you to think about:

- When you complete the chain, what can you see and why is this happening?
- What happens when you use an even number of inverters instead of odd?
- What information do we need in order to work out what frequency the Ring Oscillator is running at?
- What would happen if we reduced the supply voltage to the NAND board, for example, to 3 V? (The standard provided via USB is typically 5 V.)

Thinking about and trying to answer these questions will further your understanding of digital logic. Discuss your answers with your lab partner and other groups.

Well done **for completing all the NAND labs.**