Lab 5: Finite-State Machines

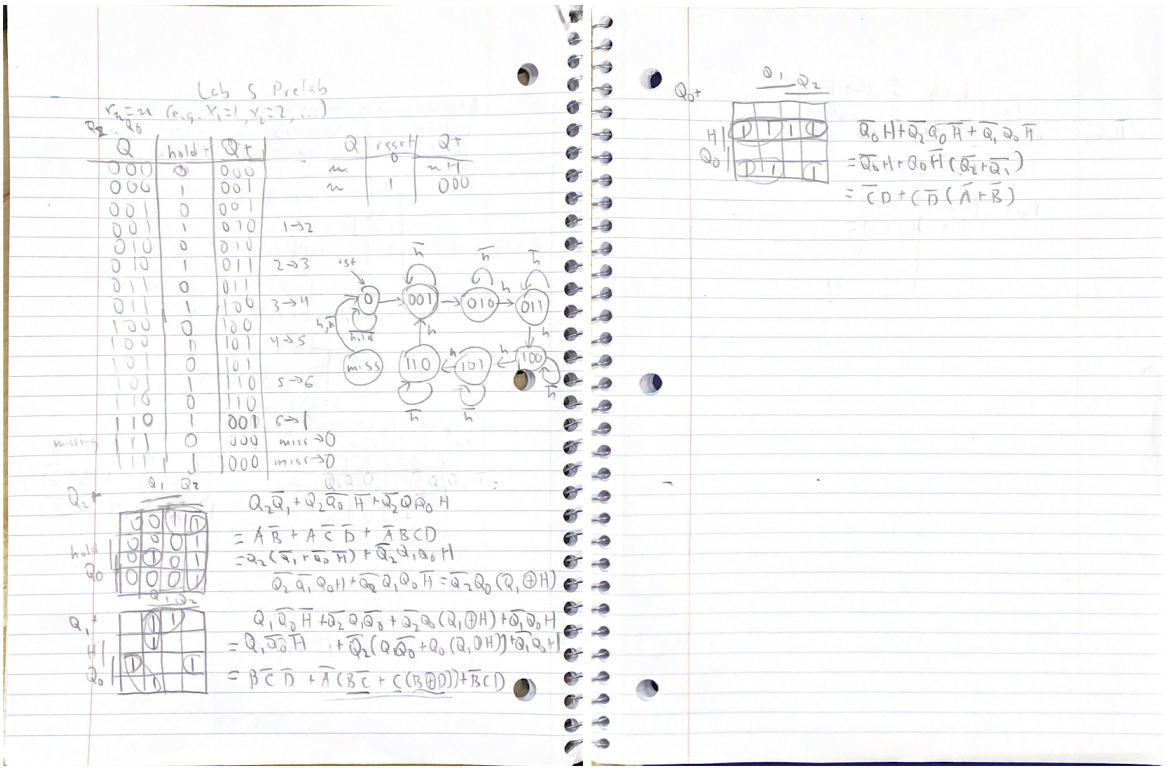
Eric Liu

Objectives

In this lab, our objective is to better understand Finite State Machines, and implement a Moore machine onto our FPGA. We will be making a “dice roller” circuit, which will roll a dice when a button is pressed and land on a number when the button is released. It can also be reset with another button.

Design and Test Procedure

Prelab:



Here, I simply assigned $X_n = n$, which means $X_1 = 1, X_2 = 2$, etc.

As you can see from the state diagram near the middle of the pages, this is a circuit that “rolls a dice”. It starts at zero and then increments through 1 to 6 before resetting to 1, and you can hit a reset button to set the counter back to 0. There’s also a small bit of logic to make sure that the counter doesn’t get stuck at state 7, since that’s not a valid state of the counter.

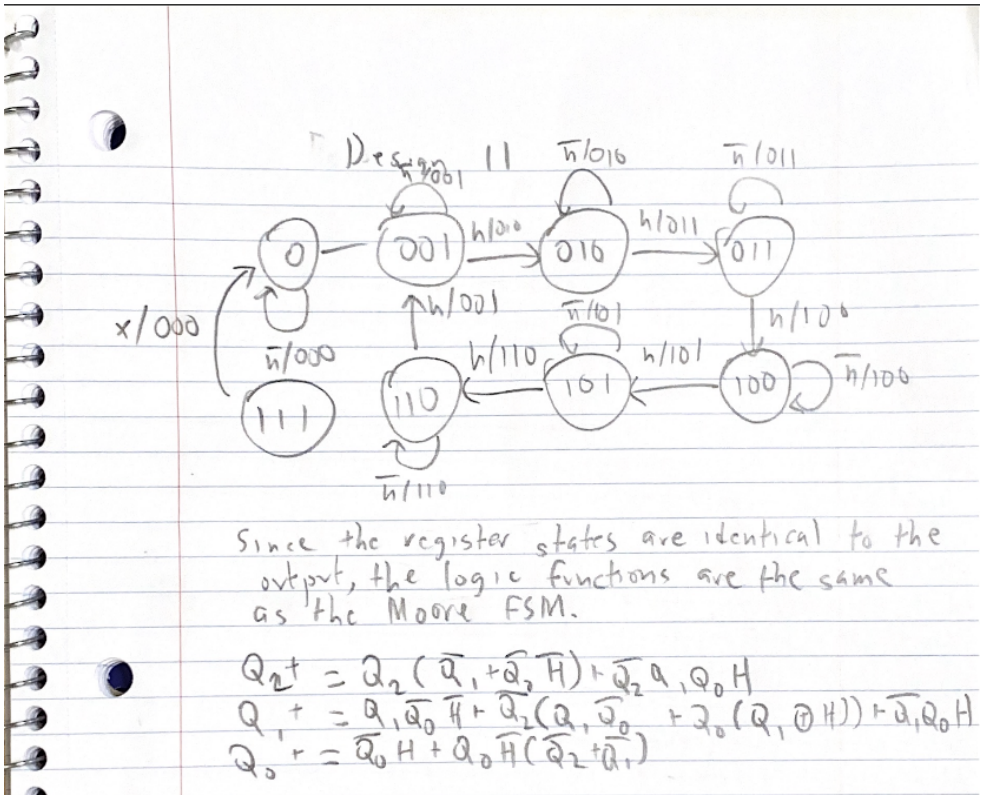
The functions from the K-maps are as follows:

$$Q_2^{+} = Q_2(\neg Q_1 + \neg Q_0 \neg H) + \neg Q_2 Q_1 Q_0 H$$

$$Q_1^{+} = Q_1 \neg Q_0 \neg H + \neg Q_2(Q_1 \neg Q_0 + Q_0(Q_1 \oplus H)) + \neg Q_1 Q_0 H$$

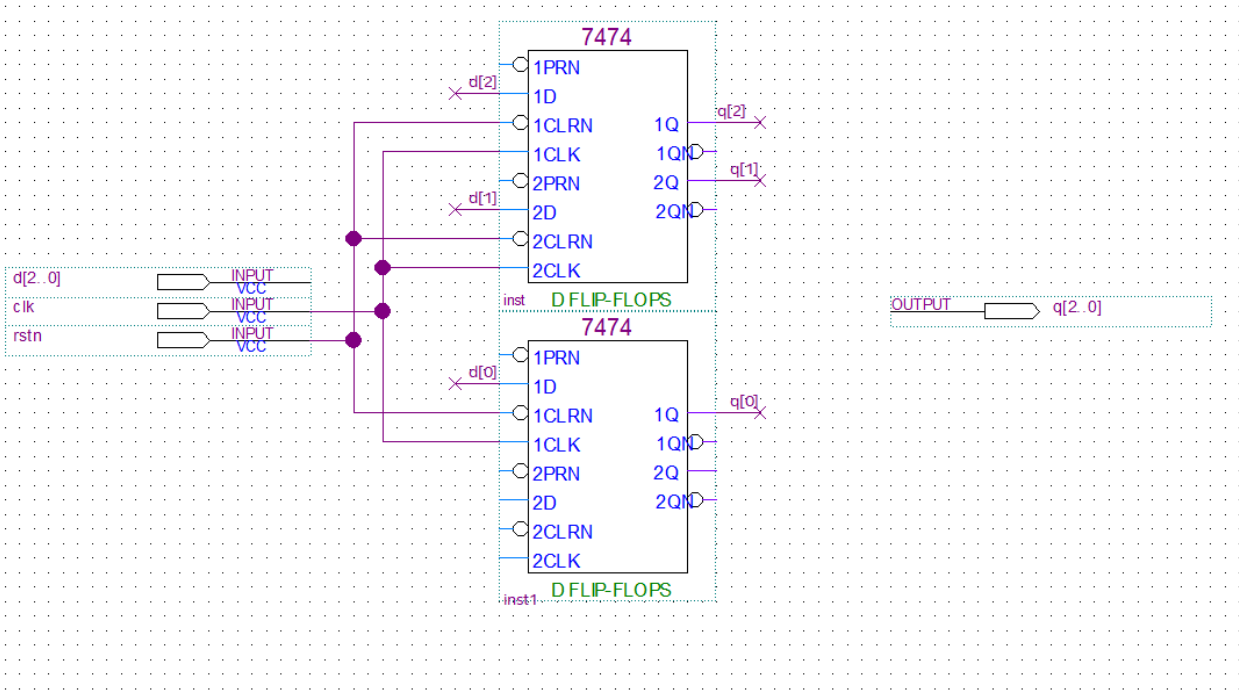
$$Q_0^{+} = \neg Q_0 H + Q_0 \neg H(\neg Q_2 + \neg Q_1)$$

Mealy machine:



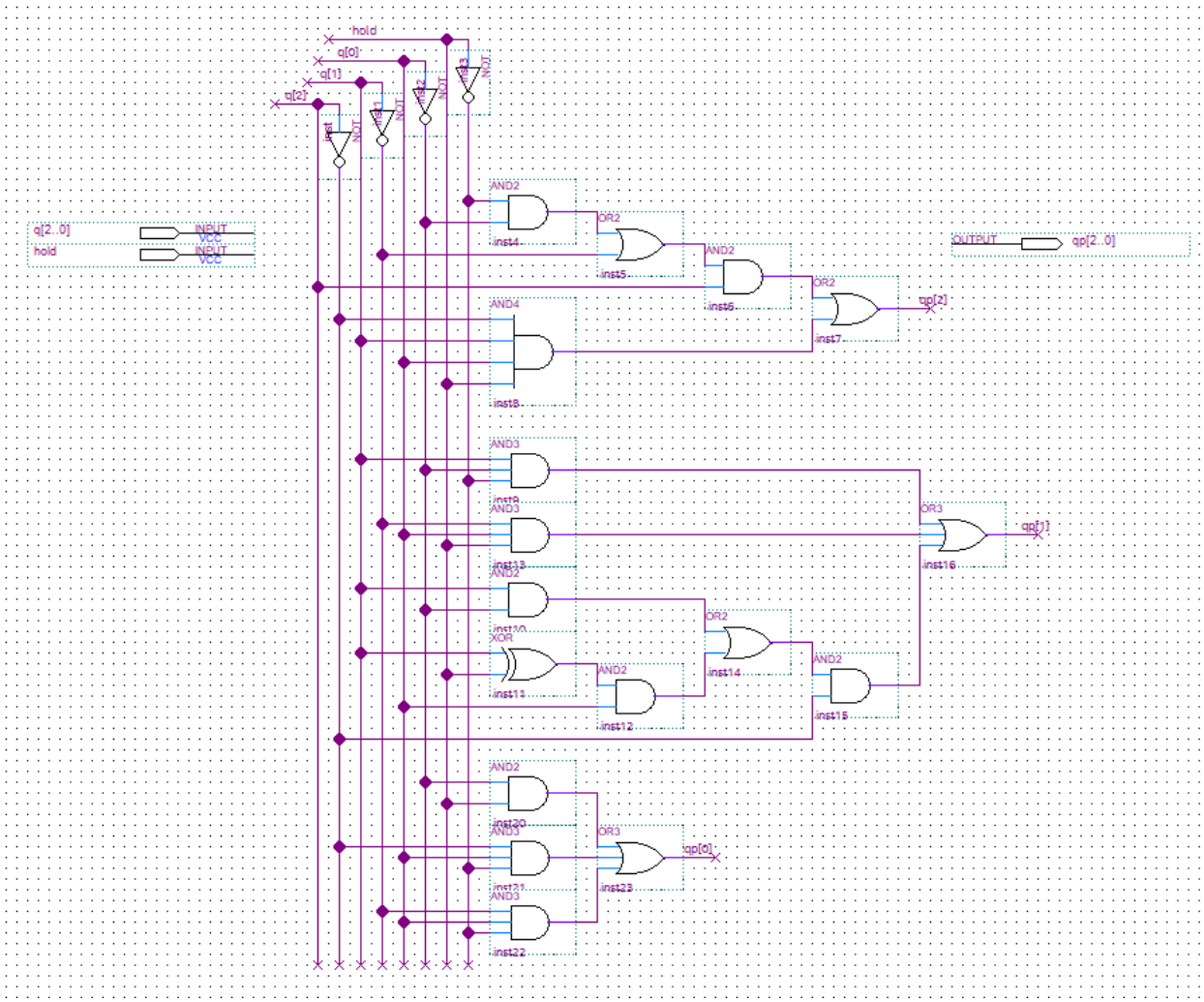
This is the Mealy machine design. You can see that the combinational logic is identical, because the output is simply fed directly from the combinational circuit. The only difference is that the output is asynchronous because it's directly wired to the combinational logic instead of being updated through the register, which is only refreshed when the clock goes from negative to positive. This has some advantages (less delay), but is bad because if the combinational logic has some glitch, it will be directly observable in the output. Here it's not that big of a deal because it would simply be an erroneous number displayed on the 7-segment display, but for a more sophisticated circuit this might be an issue because the output of this circuit might be fed into something that's very reliant on the input and it might mess up the state in the long run. More details in the Response to Questions section.

3-bit register:



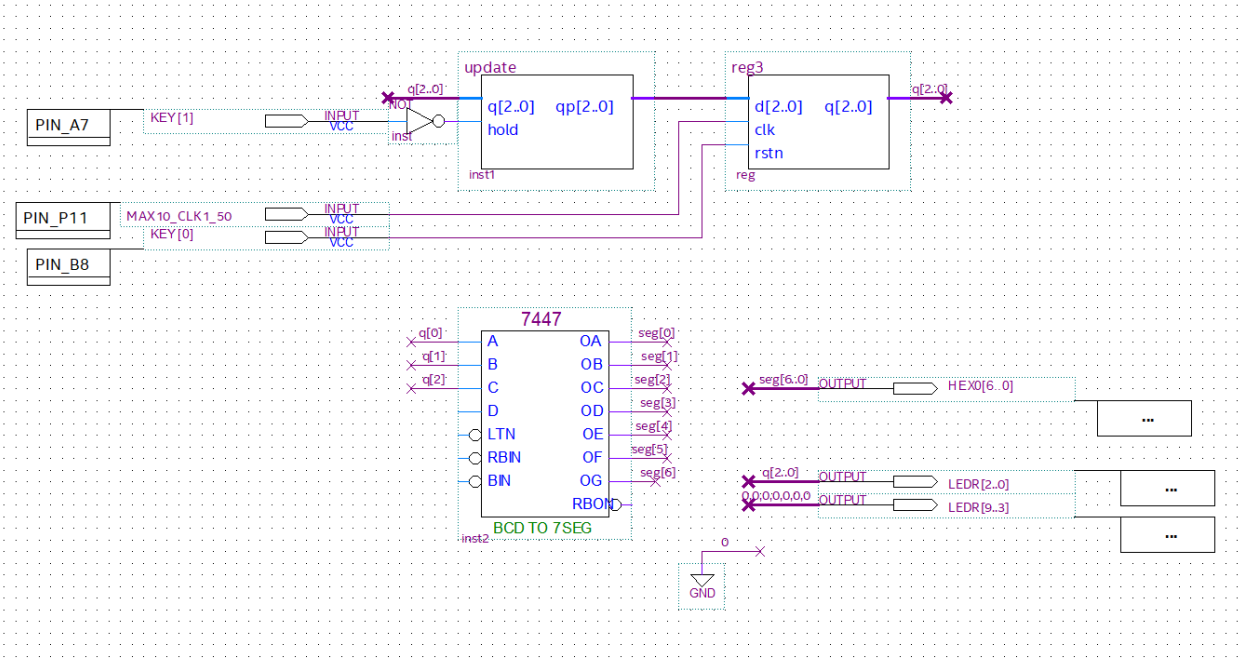
This is just 3 D flip-flops with the same `CLK` and `RSTN` inputs. Nothing special. It's used to store all of the states of the FSM, as described in the state diagram.

Update logic (Q to Q+):

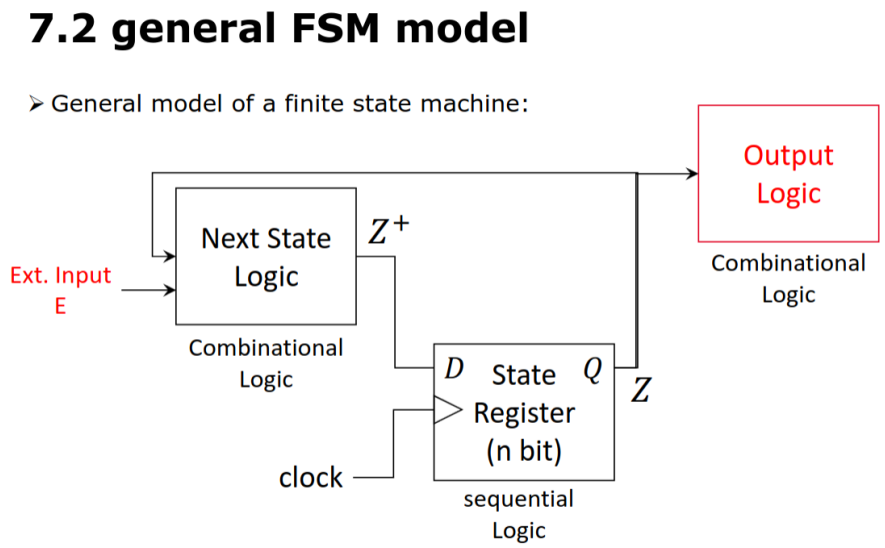


This is derived from the K-maps shown in the prelab. This contains all the update logic the counter relies on: it takes the current register state and the signal provided by the roll button, and updates the register accordingly.

Top level design:



This is the top level schematic, with the register and update logic hooked up to each other with 7-segment display and LED outputs so it's easy to debug.
I added the LED output to the register's outputs so I can see the outputs in binary format in ModelSim: it makes it a lot easier to interpret the results, as shown in the results section.
This layout is very inspired by the diagram shown in class:



I feel laying out the circuit like this makes it a lot easier to debug because it isolates bugs within the boundaries of each individual component.

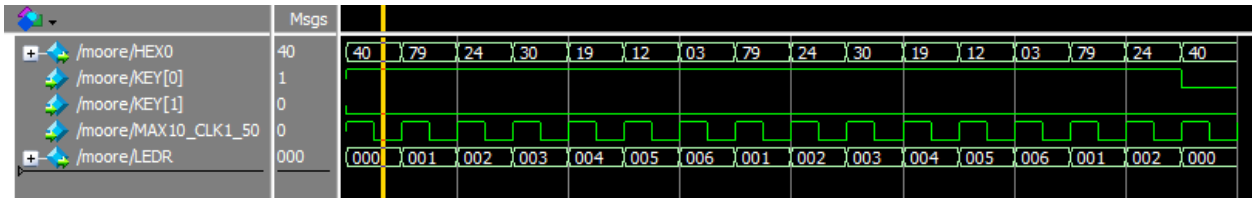
Results and Answers to Questions

Part 1 simulation script:

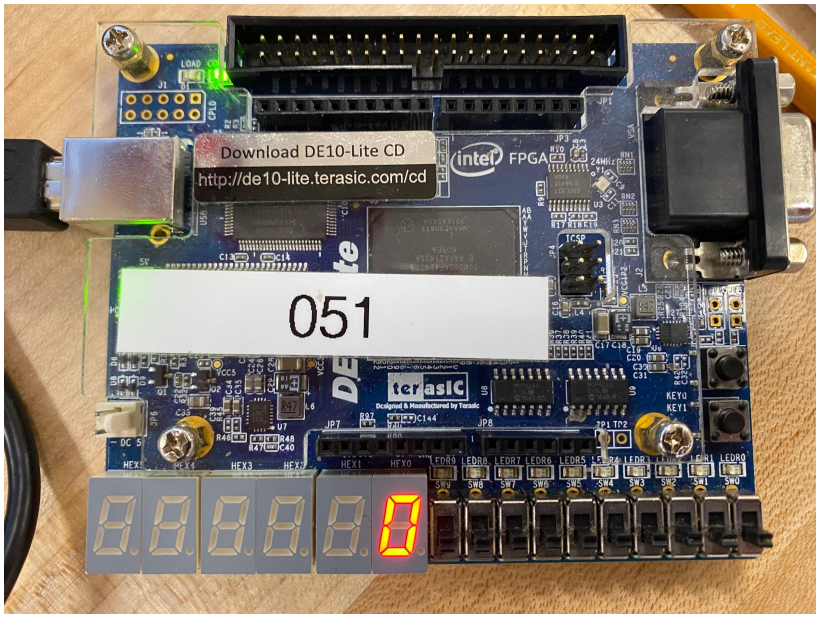
```
# Configure CLK
force -freeze sim:/moore/MAX10_CLK1_50 1 0, 0 {50 ps} -r 100
force -drive {sim:/moore/KEY[0]} 1 0
force -drive {sim:/moore/KEY[1]} 0 0

# Run 2 iterations (0 → 6, 6 → 1)
echo "Testing 2 cycles"
run 1500 ps

# Try reset
force -drive {sim:/moore/KEY[0]} 0 0
echo "Testing reset"
run 100 ps
```

This script is pretty self explanatory; it first configures the clock and inputs, and then runs a certain number of cycles to test the FSM's looping logic. Then it resets the register to make sure the reset transition works.

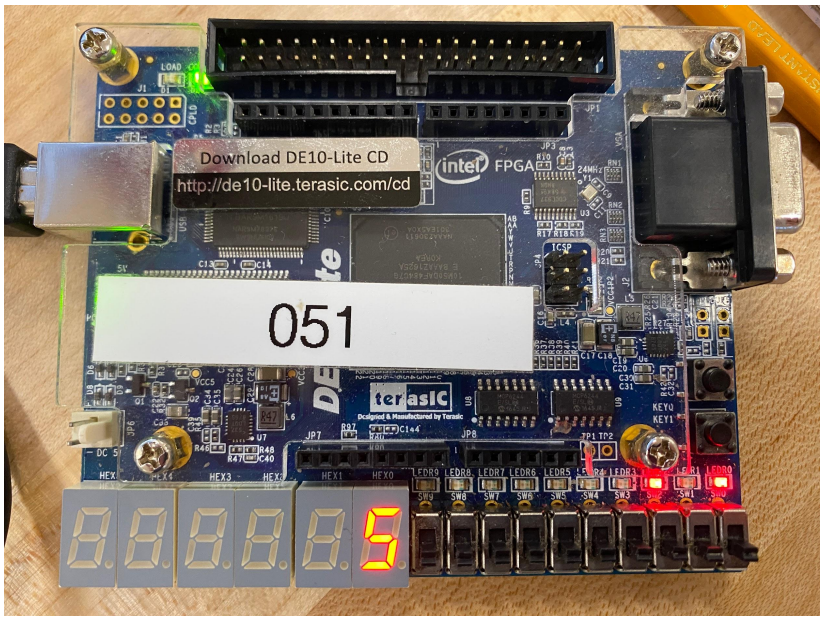
Part 1 simulation results:



This is where the LEDR output was very useful. You can see the outputs from it are much more readable than the ones from HEX0, which are harder to interpret.
This simulation makes a lot of sense. The register first starts in the reset state, then enters the loop and cycles from 1 to 6 until it stops at 2, and then I "activate" the reset key (low active) to set the register back to the reset state.



This is the initial state of the FSM: it freezes at 0 until the roll button is pressed.



This is after I released the roll button. This time it landed on 5. You can also see the LEDR inputs, which is just 5 in binary (101).

Q1: The difference between my Mealy and Moore machines are very similar; the only real difference between them is that the output is not fed through the register in the Mealy version, which makes the output susceptible to glitches but also allows it to be asynchronous. The advantage of this is that the output is slightly faster because it doesn't have to wait for a positive clock edge to be updated, but the disadvantage is that the output is very susceptible to glitches because there is no compensation if one of

the bits of the output get propagated faster than the others (for example, if Q_0^+ updates faster than Q_1^+ and Q_2^+, then you might get an output of 001 rather than 011 for a split second, because Q_1^+ and Q_2^+ are still propagating through.)

Q2: The Moore FSM that aligns the outputs with the register state would be the one with the least gates, because the output would be identical to the current state of the register. If you had to make the register only count up and the output had to map from the register state to a given number, you'd have to make another combinational component with a nonzero amount of gates, so it'd be less efficient overall.

Conclusions

In this lab, I learned how to design and implement a finite state machine, by making a dice roll circuit. I learned how to implement this as a Moore and a Mealy machine, along with learning how to properly test the circuit and use the clock input during simulation in ModelSim.

I ran into a couple issues in this lab. I accidentally set the wrong transition state between X_6 and X_1 (110 -> 000), so I had to redo my K-map for output Q_0^+ a second time. I also didn't initially understand the difference between a Moore and a Mealy machine, which slowed down my progress quite a bit.

I don't think there was much I could've improved upon my design procedure. I think perhaps the combinational update logic I wrote could've been a little more reductive and would have consisted of slightly less circuits, but I don't think it's that big of an issue and it probably would've been harder to read the schematic if I did optimize for the number of gates. My test procedure was quite streamlined because I used a Tcl script to run it, so I don't think there was much to be improved there.

The results of the lab were pretty expected. The goal of the lab was very clearly laid out and there wasn't much surprise with what my output was, other than messing up the state diagram once. Overall though, this was very educational in how to approach designing and testing a FSM, which will be very important when designing the logic for our next (and final) lab.