ECE 410 Lab 2: Inverter Simulation

Authors: Yaaqoob Choulli, Rowan Rasmusson

Student ID: 1752634, 1757167

Course: ECE 410

Lab Section: D21

Date Performed: October 22, 2025

Abstract:

This lab includes a dive into how sequential circuits can be used and specifically their application to making an FSM. The first part of the lab consisted of creating a 4-bit countdown timer; a basic FSM to demonstrate controlling the state of a sequential circuit during the countdown of a number and being able to load and countdown from different values. The second part was a demonstration of an application of sequential circuits in pseudo-random number generation. We implemented a Tauseworthe PRNG that used a clock-driven register, some combination logic and shift operations to produce a random 32-bit number. Lastly, a larger FSM was implemented to simulate a secure element chip in charge of holding cryptographic keys and protecting against being tampered with. All these designs performed successfully on the Zybo Z7-10, with state transitions being tested to completion.

Design:

Part1:

For part one, our design consisted of two processes: one for monitoring the asynchronous reset and updating the necessary values every clock cycle, and the second responsible for transitioning between states and controlling the display output.

A specification was given to have the default loaded value be 10, so we implemented this, assuming that the user doesn't want to save their loaded value when resetting the device. One design decision was our choice to display the loaded value during the IDLE and LOAD states. This lets the user know what value is about to be counted down from if they assert START. Another decision was to return to the IDLE state after loading a value. This was to let the user know that the loaded value is ready to be counted down from, and the logic is not actively loading a value into the register. Lastly, a DONE state was used even though it didn't end up having any use for this lab. We imagined that in another application, it would be useful for the user to know when the counter is done counting down. But this was not needed for the specifications specifically for part 1. (A picture of the FSM diagram can be found in the appendix.)

Part 2:

The Tauseworthe PRNG was a simple sequential circuit involving shift operations, XOR and AND logic. The image below shows the high-level block diagram of a single generator. The outputs of these generators were XOR'd together to make a 32-bit output. Our actual implementation of the generator in top_taus.vhd is quite simple and requires straightforward logic. One important design decision that we made was making the parameters S1, S2, S3 and C **generics**. Having these as **generics** in the VHDL language meant that we would only have to implement one Behavioural entity, but just pass different arguments to the GENERIC MAP when instantiating the three different Tauseworthe generator instances in the taus88.vhd file. Finally, an extra process was needed in the taus88.vhd file to XOR the outputs of the three instantiated PRNGs in the final Structural architecture. (A picture of the entire structure block diagram can be found in the appendix, from prelab.)

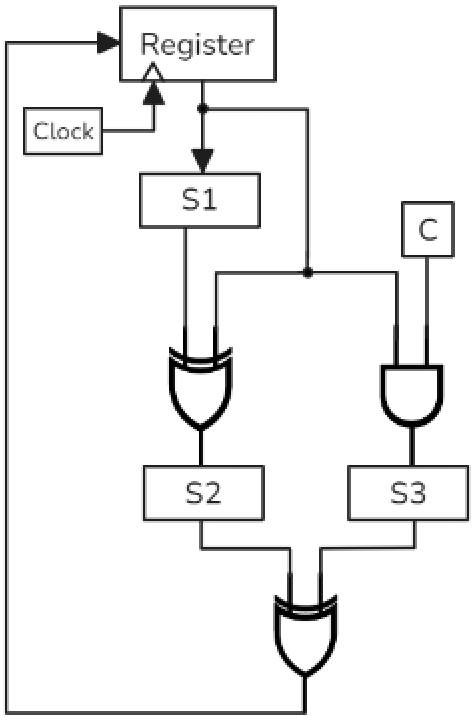


Figure 1: Tausworthe Generator

Part 3:
In part 3 of lab 2, we implemented a finite state machine to replicate that of a TROPIC 01 secure element chip partially. The 5 valid states for our secure element FSM were STARTUP, IDLE, SLEEP, SECURE_CHANNEL, and ALARM. Like Part 1, the FSM was divided into two processes: one for updating the state on each clock cycle and allowing for an asynchronous reset, and the second for defining the valid state transitions allowed in the state machine.

A deviation we took from the lab handout was making our sleep mode RGB colour completely off instead of white. We did this as a lesson from Part 1 since white is the colour the LED shows when it is rapidly switching between states/colours. To easily differentiate between SLEEP and this fast switching of states, we chose to make sleep be no colour on the LED.

We chose to include the reset input as part of part 3 to make testing easier, since according to the design specifications in the lab handout, there is no way of getting out of ALARM once triggered.

See the appendix for part 3 state diagram.

Part 4:
The purpose of part 4 was to combine the countdown timer built in part 1 with the FSM built in part 3. Two main additional functions that part 4 is to demonstrate are the ability to wake up

after being in the SLEEP state for 15 seconds, and the ability to blink the red LED in the ALARM state at 1 Hz. The former uses the countdown timer FSM from part 1, the latter uses the 1 Hz clock divider that was also used to count down in part 1.

Linking these two FSMs together required adding an extra *done_out* output to the countdown_timer_fsm and a *start_cd* output from the secure element FSM. These signify when the timer's count has reached zero and when the timer should start counting down from a default number, respectively. The *start_cd* was sent to the *start_btn* input of the countdown_timer_fsm whenever the secure element FSM was put into the SLEEP state. *Cd_done* was asserted when the count was done, and the secure element FSM took this to go back into the IDLE state.

We chose to have the secure element FSM wake up into the IDLE state instead of the STARTUP state because this state is the most responsive to the widest variety of inputs. Whether the user wants to go to SECURE_CHANNEL, ALARM, SLEEP again or back to STARTUP. This state offers the greatest flexibility; thus, we thought it would be the best choice to have the chip wake up into after sleeping for 15 seconds.

<u>Simulation:</u>

Part 1:

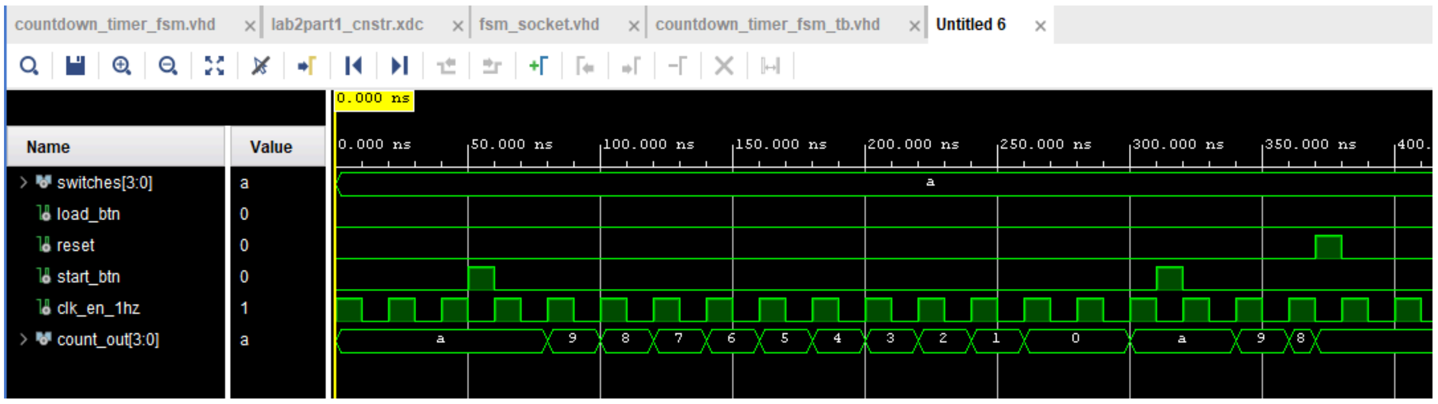


Figure 2: This simulation for part 1 shows an initial run of 10 seconds; 10 being the default if no value is loaded using the switches. In this testbench run, start_btn is the only input that is asserted to make sure the default count works. The asynchronous reset can also be seen working since it interrupts a second countdown that is started at around 300ns. Count_out[3:0] is the output displayed on the SSD while the FSM is active.

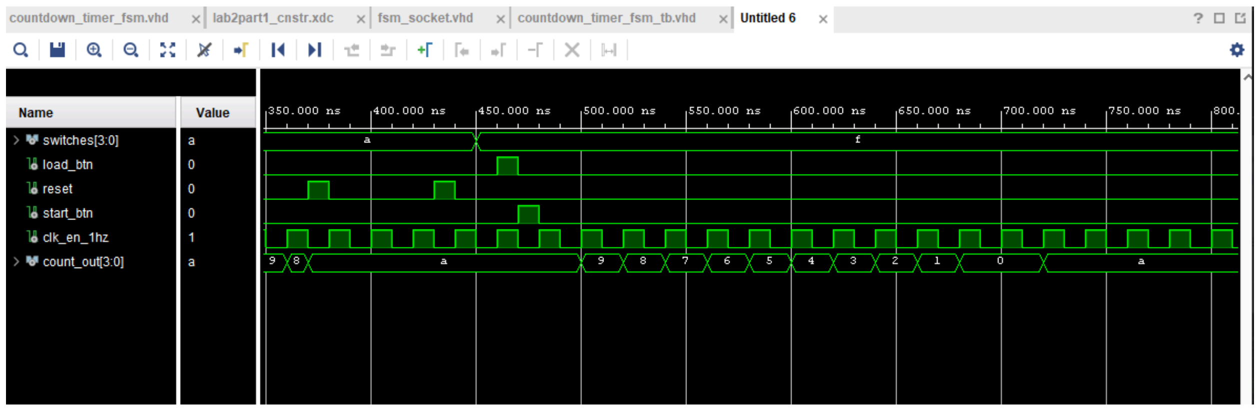


Figure 3: This figure shows a later simulation with a small error. The load_btn doesn't load 'f' (15) into the FSM before the start_btn is pressed. Instead, it starts counting down from 10. This was a discrepancy between our simulation and the Zybo Z7 board. The test cases chosen for part 1 were to make sure the asynchronous reset worked, since this is an important part of the functionality. Also, to make sure the default value was used when no value was loaded from the switches. Lastly, the test case to test the loading functionality was chosen because it should demonstrate the ability to choose different values to be chosen for the start of the countdown. Even though it failed in simulation, it was successful on the board.

Part 2:

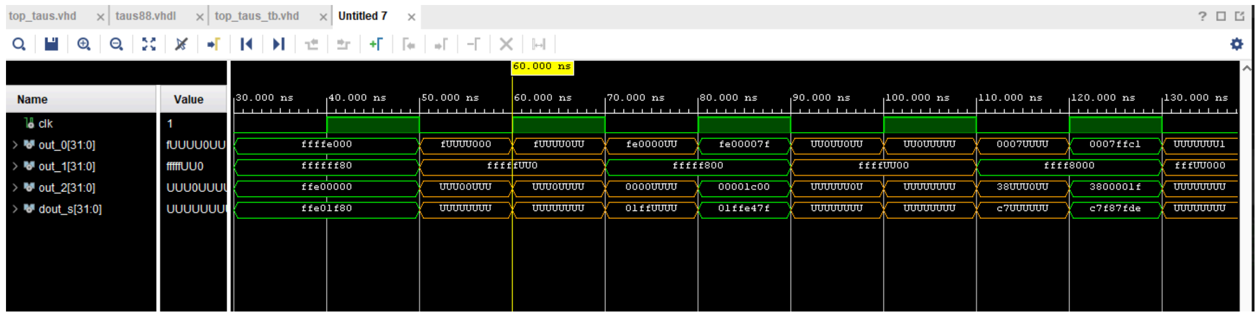

top_taus.vhd
taus88.vhdl
top_taus_tb.vhd
Untitled 7
60.000 ns
Name
Value
clk 1
out_0[31:0] fUUUU0UU
out_1[31:0] ffffUU0
out_2[31:0] UUU0UUUU
dout_s[31:0] UUUUUUU
30.000 ns
40.000 ns
50.000 ns
60.000 ns
70.000 ns
80.000 ns
90.000 ns
100.000 ns
110.000 ns
120.000 ns
130.000 ns
ffffe000 fUUUU000 fUUUU0UU fe0000UU fe00007f UU0UU0UU UU0UUUUU 0007UUUU 0007ffc1 UUUUUUU1
ffffff80 ffffUU0 fffff800 ffffUU00 ffff8000 fffUU000
ffe00000 UUU00UUU UUU0UUUU 0000UUUU 00001c00 UUUUUU0U UUUUUUUU 38UUU0UU 3800001f UUUUUUUU
ffe01f80 UUUUUUUU UUUUUUUU 01ffUUUU 01ffe47f UUUUUUUU UUUUUUUU c7UUUUUU c7f87fde UUUUUUUU

Figure 4: This simulation of our Tausworthe PRNG shows the dout_s[31:0] signal being determined by out_0, out_1 and out_2 being xor'd together. Even though the correct output was produced both on the board and in simulation every 4 clock cycles, some undefined signals were seen in the simulation between outputs. The reason for this is discussed in the results section of the report. The test cases for this PRNG were just to run the clock and see if the expected 'random' numbers matched the ones given in the test file for part 2, making testing quick but effective.

Part 3:

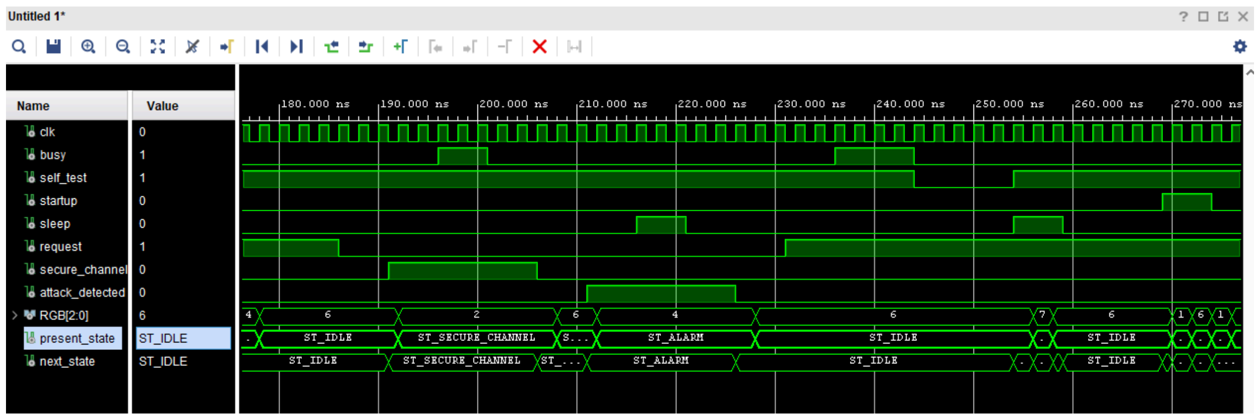


Figure 5: This simulation for part 3 shows a number of successful state transitions that follow expected behaviour. Some of the transitions shown are IDLE -> SECURE_CHANNEL when the secure_channel input was asserted. Then IDLE -> ALARM when *attack_detected* was asserted. ALARM was shown to hold its state when the *sleep* input was asserted during the ALARM state. Additionally, ALARM was allowed to go back to IDLE for simulation because we needed a way to get out of ALARM to test other transitions. Next, IDLE was shown to hold state when *request* was asserted and a brief transition to SLEEP, then back to IDLE when both *sleep* and *request* were asserted. Lastly, *startup* is asserted to go back to STARTUP (RGB=1), but since *self_test* was 1 and *busy* was zero, it toggled between STARTUP and IDLE while *startup* was 1. Pictured here are just some of the test cases chosen, though we made sure our testbench file for parts 3 and 4 covered all possible state transitions to ensure the robustness of our secure element FSM.

Part 4:

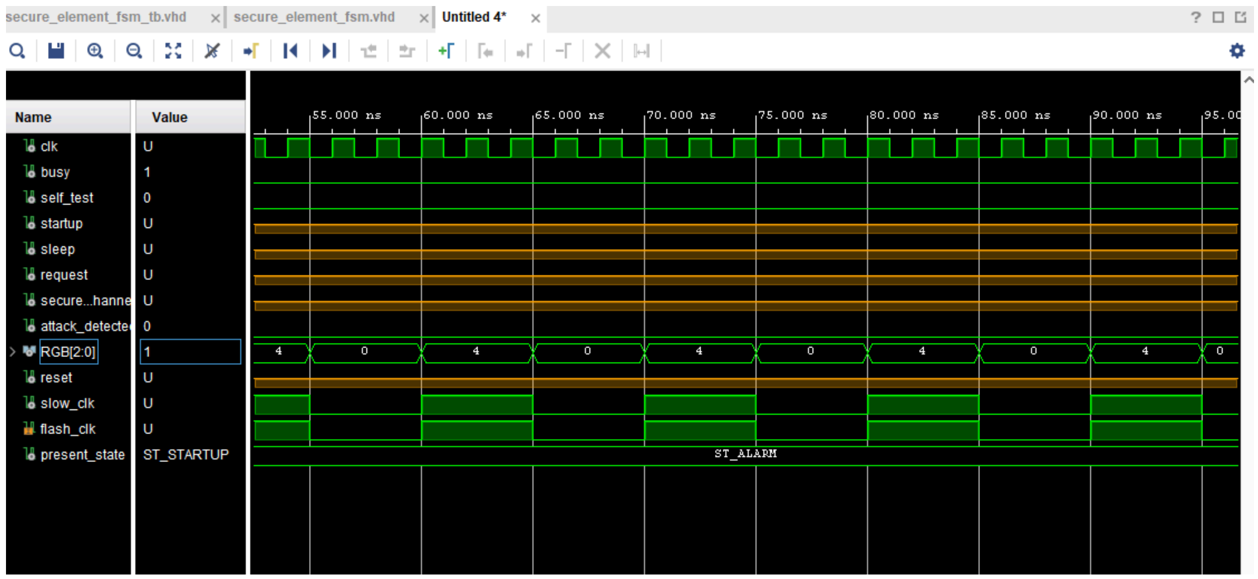


Figure 6: This simulation for part 4 demonstrates the functionality of the blinking red LED during the ALARM state of the secure element FSM. As can be seen, the RGB toggles between 4 (100 = red) and off (0). This is triggered during the ALARM state that is seen in the *present_state*

signal. The uninitialized values were signals that were insignificant for this portion of the testbench. The test cases for part 3 for commented out at this moment.

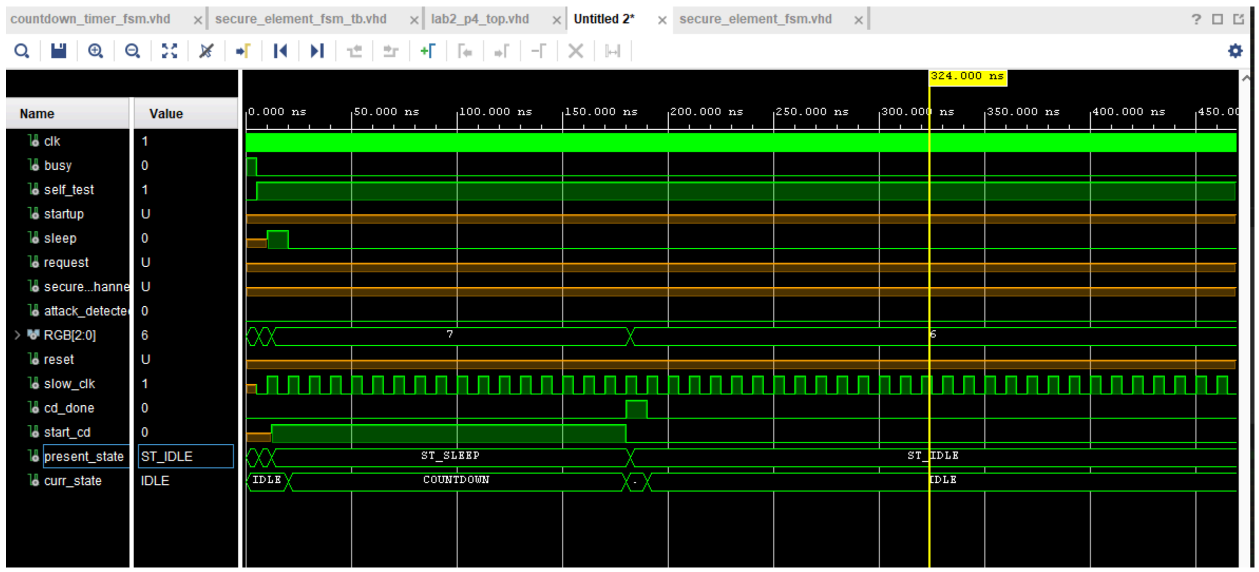


Figure 7: The test case demonstrated here is to show the functionality of the secure element FSM waking up after 15 seconds in the SLEEP state. The *slow_clk* simulates the 1 Hz clock used when the design is implemented on the real board. When 15 clock cycles are done, the countdown_fsm, which was a slightly modified version of part 1, asserts the *cd_done* signal, telling the secure element that it's time to wake up from the SLEEP state and go to IDLE where it can handle more transitions.

<u>Results:</u>

Part 1:

Overall, the lab was quite successful, whether for the individual parts or when they were hooked up together. Starting with part 1, we successfully designed the counter FSM (in the prelab), then translated it into a VHDL design. We did run into slight issues with the simulation results, which can be seen in Figure Y. When we synthesized then implemented the design on the FPGA we found that the results were all working as intended. The design used to simulate is the same one used to physically test on the board yet we had discrepancies. We got some input from our lab instructor Antonio, where he described different synthesization algorithms that can be used. We theorized that this was the culprit for the differences.

Apart from the results, this section of the lab did come with a few difficulties. We had a lot of trouble with our sensitivity list. Initially, we placed the CLK signal in our sensitivity list for our pure combinational logic process. This took us a while to deduce and came with some frustrations. On the other hand when we conducted part 3/4, it was primarily smooth sailing since a lot of our problems were already faced before in this first portion.

Part 2:
The second portion of the lab involved simulating and implementing our Tausworthe PRNG design. Results for this were similar to part 1, where overall the design worked flawlessly when implemented on the board, but had small discrepancies with simulation. Looking at Figure 4, we can see that there are some uninitialized bits going through the results. The reason we deduced was that our testbench did not initialize signals prior to running the simulation. It is important to note that this had no effect on the overall output of dout_s on the physical board; for that reason, we chose to neglect these discrepancies.

Part 3-4:
Parts 3 and 4 are the final portion of the lab. For both these parts, we achieved success in the simulation and the overall result. For part three, we designed and implemented a secure FSM system. It can be seen through our simulation (Figure 5) that we were able to test many different possibilities and achieve success. When we implemented the desig,n we achieved similar success. In the next part (Part 4), we iterated on our initial design. With respect to the results, we were able to do a regression test using our part three tests. This resulted ina a successful simulation and successful board implementation. In addition to the regression, we tested the new features of the FSM; once again, we found working successful results in both the simulation and implementation.

<u>Conclusion:</u>

This lab was split into 4 parts, all of which taught us how we can use VHDL to create sequential circuits and FSMs. In the first part, we were tasked with designing and implementing a 4-bit countdown timer. We had some difficulty regarding the separation of sequential and combinational logic and simulation discrepancies; however, in the end, we were able to come up with a successful implementation on the physical board. In the second part of the lab, the design of a Tauseworthe PRNG, we acquired similar successful results to part 1. The third part of the lab was to create a secure element FSM. Using our experiences from part 1 and 2, we had complete success in both implementations and simulation. This was the same case for part 4, where we combined parts 1 and 3.

As mentioned above, the main challenges had to do with understanding how to structure sequential logic and, in turn, the VHDL language itself. However, we have noticed that our understanding of the language has greatly improved by the end of the lab.

If we were to run this test again, we would do additional planning prior to beginning the lab. For part 1, we felt underprepared. For the upcoming lab, we would like to have a structure for the VHDL code prior to starting. Another important improvement we would make would be to test features one at a time. For part ,1 we wrote the entire FSM, then began to test. This approach was also one of the reasons we ran into issues. Moving forward, we will do one part then test it and iterate. (We did this for parts 3,4)

Ultimately, this lab reinforced our understanding of VHDL, FSMs and strengthened our understanding of digital system design and implementation.

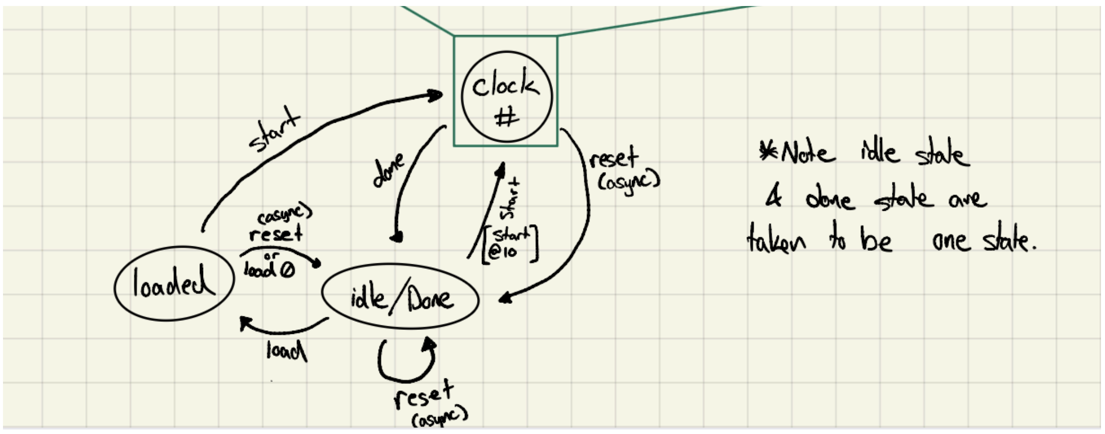
References:

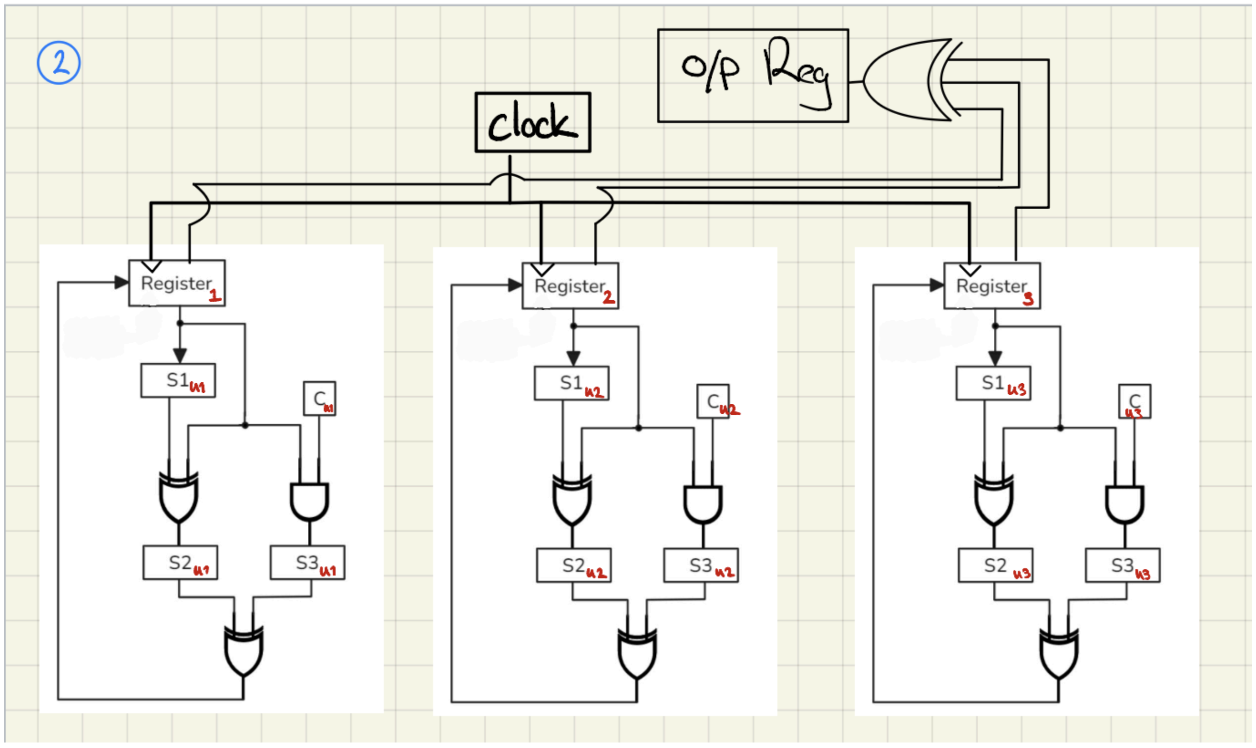
[1] Provided lab documentation
[2] Inclass notes
[3] Lab Instructor and TAs

Team Contribution:

- The pre-labs were done individually
- The labs were conducted together, both of us were writing code, referencing the notes, writing test cases and verifying test cases/functionality.
- The lab report was split evenly, and all sections were a combination of both of us.

Apendix:


clock #
start
done
start
[start @10]
reset (async)
(async) reset or load @
loaded
idle/Done
load
reset (async)
*Note idle state & done state are taken to be one state.


2
O/P Reg
clock
Register 1
S1 u1
C u1
S2 u1
S3 u1
Register 2
S1 u2
C u2
S2 u2
S3 u2
Register 3
S1 u3
C u3
S2 u3
S3 u3