CS 2110 Homework 4

Joshua Viszlai, Sam Gilson, Cem Gokmen

Fall 2018

Contents

1 Overview

In this assignment, you will learn the basics of sequential logic, and create both a one-hot state machine and a reduced state machine.

2 Part 1: RS Latch, Master-Slave D Flip-Flop, and Register

For this part of the homework you are going to make your own Register, from the ground up. All of the circuits in this part will be in the `latches.sim` file.

2.1 RS Latch

Open up the the RS Latch subcircuit in the `latches.sim` file. For this part you will need to create an RS Latch. **Note**: You only need 1 output, so you may disregard the inverse output of a normal RS Latch.

You can find information about this circuit in the Patt & Patel textbook, pages 65 and 66.

2.2 Master-Slave D Flip-Flop

Open up the D Flip-Flop subcircuit. For this part you will need to create a Master-Slave D Flip-Flop using the RS Latch you just created. **Note**: Your implementation of this Master-Slave D Flip-Flop should use only change on the **rising edge**, i.e, the state of the D Flip-Flop should only be able to change at the exact moment when the CLK (Clock) input goes from 0 to 1.

Consider implementing a Gated D Latch subcircuit first.

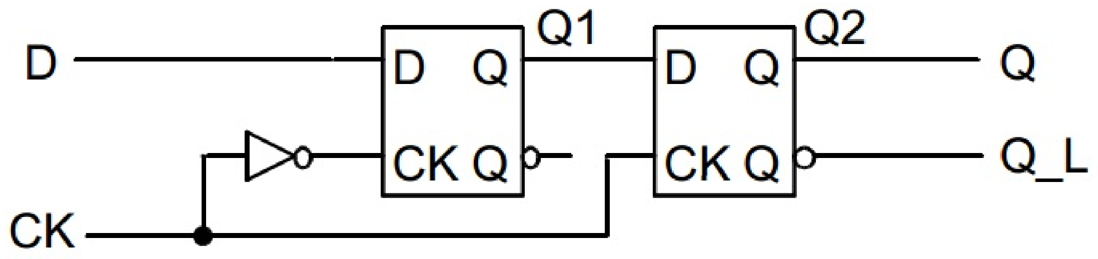

Figure 1: An example of the Master Slave D Flip Flop, with the boxes representing Gated D Latches

You can find information about the Gated D Latch and this circuit in the Patt & Patel textbook, pages 65 and 66.

2.3 Register

Open up the Register subcircuit. For this part you will need to create a **4-bit** Register. This Register also needs to use edge-triggered logic. **Hint**: Split the 4-bit input and utilize the D Flip-Flops you just created.

3 Part 2: Finite State Machine

For this part of the homework, you will implement the finite state machine behind a parking garage gate.

3.1 The State Machine

We are modeling the finite state machine behind a boom barrier (think gated parking garage entrances).

3.1.1 Scenario

1. State: We start with the barrier closed, there is no car, and the SCAN YOUR BUZZCARD light is on.
2. Event: A car approaches the machine and the driver scans their card.
3. State: The barrier opens and stays open.
4. Event: The driver drives under the gate, the sensor under the gate starts showing that there is a car underneath.
5. State: The barrier stays open, waiting for the car to pass.
6. Event: The driver clears the gate, and the sensor under the gate starts showing that there is not a car underneath.
7. Our state machine loops back to the beginning, with the gate closed and light on.

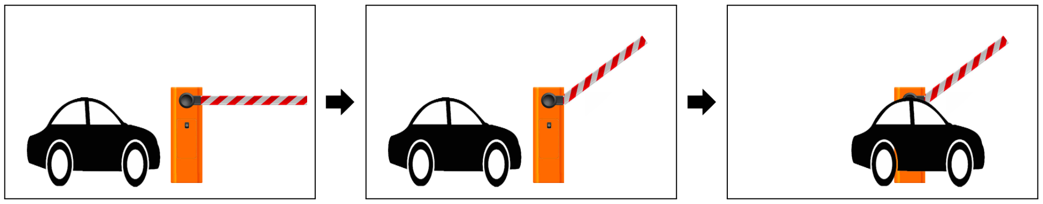
Figure 2: Our state machine with its 3 states in order.

3.1.2 States

Our state machine has 3 states: **State 0: Gate Closed**, **State 1: Gate Open**, and **State 2: Car Under Gate**. Our state machine begins with *State 0: Gate Closed*.

3.1.3 Inputs

- G = Valid card scan from card scanner (1 when a valid card scan has been made, 0 otherwise)
- F = Gate sensor: whether or not there is a car passing under the gate (1 when a car is underneath the gate, 0 otherwise)

3.1.4 Outputs

- A = *Scan Card Now* light at the kiosk
- B = Signal to the motor to keep the gate open
- C = Signal to the motor to keep the gate closed

Note that B and C are the inverse of each other and there's no practical reason why you'd do this in real life. They exist just for the sake of having more outputs.

3.1.5 State - Output Mappings

- State 0: Scan card light and closed garage door
- State 1: Open garage door
- State 2: Open garage door

3.1.6 Transitions

1. If we are in state 0 and $G == 1$ (a card was read) we transition to state 1.
2. If we are in state 1 and $F == 1$ (a car is detected underneath) we transition to state 2.
3. If we are in state 2 and $F == 0$ (the detected car is gone) we transition to state 0.

Note that these rules don't include every possible combination of inputs and current state. You should interpret these rules as sufficient - for example, if we are in state 0 and $G == 1$, we should transition to state 1 regardless of the value of F. Similarly, all input combinations not containing $G == 1$ at this state should result in staying in this state.

3.1.7 Finite State Machine Diagram

Here is a diagram of our state machine containing all of the information listed above:

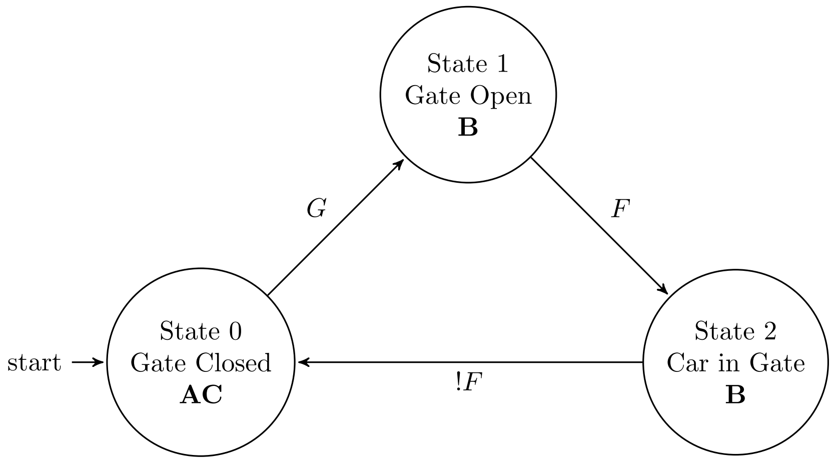

Figure 3: The State Machine Diagram for our State Machine

3.2 One-Hot Encoding Implementation

For this part of the homework you will need to implement our state machine in CircuitSim using a **One-Hot Encoding**. What this means is that we use 1 bit per state to represent our current state. So with our 3 states, we will need 3 bits. **It is important that this machine stays one-hot: there should be at most one bit set to 1 in the register at any time**.

Use the following mappings for each state in your One-Hot state machine:

- State 0: `001`
- State 1: `010`
- State 2: `100`

Make this circuit in the One-Hot State Machine subcircuit in the `fsm.sim` file. Also, notice along with G and F there are 2 other inputs: `CLK` is the clock and will be used to simulate the clock of the circuit cycling on and off, and `RESET` should reset your state machine.

Your state machine, when the circuit is initially started or when the reset button is pushed, will return to the invalid 000 state. You need to handle this case to make it go to State 0, which is our start state.

3.3 Reduced Encoding Implementation

For this part of the homework you will need to implement our state machine in CircuitSim using a **Reduced Encoding**. This means we will be using two bits to represent the state instead of the three bits we used in the One-hot case.

Each state will be represented by its number in two-bit binary: State 0 corresponds to 00, **State 1 to** 01, **and State 2 to** 10. **The** 11 **encoding is unused.**

3.3.1 Worksheet

In order to reach this encoding, you need to prepare a truth table, Karnaugh maps, and finally reach the final reduced expressions that you will use for the gates in your circuit. For this purpose, we have prepared a worksheet that you will find in this archive that you must complete. This worksheet will be scanned and submitted on Gradescope, and its contents will be written on Canvas for autograding. Please follow instructions on the worksheet for more information. **You MUST complete the worksheet.**

IMPORTANT! SUBMIT YOUR WORKSHEET CONTENTS ON CANVAS TO SEE THAT YOU HAVE GOTTEN IT RIGHT BEFORE CONTINUING.

3.3.2 Circuit

Now that the worksheet is complete and you have verified your results on Canvas, it's time to implement this circuit in the Reduced State Machine subcircuit in the `fsm.sim` file. Also, notice along with G and F there are 2 other inputs: `CLK` is the clock and will be used to simulate the clock of the circuit cycling on and off, and `RESET` should reset your state machine.

Since 00 **is the default state, no additional logic for the resetting is required in this case.**

4 Deliverables

Please make sure the following items have been submitted in the correct locations:

1. **Worksheet contents** ⇒ Canvas, submission details on Worksheet
2. **Worksheet scan** ⇒ Gradescope, *Homework 4 Worksheet Scan* assignment
3. `fsm.sim` and `latches.sim` ⇒ Gradescope, *Homework 4 Circuits* assignment

Be sure to check your score to see if you submitted the right files, as well as your email frequently until the due date of the assignment for any potential updates.

5 Rules and Regulations

5.1 General Rules

1. Starting with the assembly homeworks, any code you write must be meaningfully commented. You should comment your code in terms of the algorithm you are implementing; we all know what each line of code does.
2. Although you may ask TAs for clarification, you are ultimately responsible for what you submit. This means that (in the case of demos) you should come prepared to explain to the TA how any piece of code you submitted works, even if you copied it from the book or read about it on the internet.
3. Please read the assignment in its entirety before asking questions.
4. Please start assignments early, and ask for help early. Do not email us the night the assignment is due with questions.
5. If you find any problems with the assignment it would be greatly appreciated if you reported them to the author (which can be found at the top of the assignment). Announcements will be posted if the assignment changes.

5.2 Submission Conventions

1. All files you submit for assignments in this course should have your name at the top of the file as a comment for any source code file, and somewhere in the file, near the top, for other files unless otherwise noted.
2. When preparing your submission you may either submit the files individually to Canvas/Gradescope or you may submit an archive (zip or tar.gz only please) of the files. You can create an archive by right clicking on files and selecting the appropriate compress option on your system. Both ways (uploading raw files or an archive) are exactly equivalent, so choose whichever is most convenient for you.
3. Do not submit compiled files, that is .class files for Java code and .o files for C code. Only submit the files we ask for in the assignment.
4. Do not submit links to files. The autograder does not understand it, and we will not manually grade assignments submitted this way as it is easy to change the files after the submission period ends.

5.3 Submission Guidelines

1. You are responsible for turning in assignments on time. This includes allowing for unforeseen circumstances. If you have an emergency let us know **IN ADVANCE** of the due time supplying documentation (i.e. note from the dean, doctor's note, etc). Extensions will only be granted to those who contact us in advance of the deadline and no extensions will be made after the due date.

2. You are also responsible for ensuring that what you turned in is what you meant to turn in. After submitting you should be sure to download your submission into a brand new folder and test if it works. No excuses if you submit the wrong files, what you turn in is what we grade. In addition, your assignment must be turned in via Canvas/Gradescope. Under no circumstances whatsoever we will accept any email submission of an assignment. Note: if you were granted an extension you will still turn in the assignment over Canvas/Gradescope.

3. There is a 6-hour grace period added to all assignments. You may submit your assignment without penalty up until 11:55PM, or with 25% penalty up until 5:55AM. So what you should take from this is not to start assignments on the last day and plan to submit right at 11:54AM. You alone are responsible for submitting your homework before the grace period begins or ends; neither Canvas/Gradescope, nor your flaky internet are to blame if you are unable to submit because you banked on your computer working up until 11:54PM. The penalty for submitting during the grace period (25%) or after (no credit) is non-negotiable.

5.4 Syllabus Excerpt on Academic Misconduct

Academic misconduct is taken very seriously in this class. Quizzes, timed labs and the final examination are individual work.

Homework assignments are collaborative, In addition many if not all homework assignments will be evaluated via demo or code review. During this evaluation, you will be expected to be able to explain every aspect of your submission. Homework assignments will also be examined using computer programs to find evidence of unauthorized collaboration.

What is unauthorized collaboration? Each individual programming assignment should be coded by you. You may work with others, but each student should be turning in their own version of the assignment. Submissions that are essentially identical will receive a zero and will be sent to the Dean of Students' Office of Academic Integrity. Submissions that are copies that have been superficially modified to conceal that they are copies are also considered unauthorized collaboration.

You are expressly forbidden to supply a copy of your homework to another student via electronic means. This includes simply e-mailing it to them so they can look at it. If you supply an electronic copy of your homework to another student and they are charged with copying, you will also be charged. This includes storing your code on any site which would allow other parties to obtain your code such as but not limited to public repositories (Github), pastebin, etc. If you would like to use version control, use github.gatech.edu

5.5 Is collaboration allowed?

Collaboration is allowed on a high level, meaning that you may discuss design points and concepts relevant to the homework with your peers, share algorithms and pseudo-code, as well as help each other debug code. What you shouldn't be doing, however, is pair programming where you collaborate with each other on a single instance of the code. Furthermore, sending an electronic copy of your homework to another student for them to look at and figure out what is wrong with their code is not an acceptable way to help them, because it is frequently the case that the recipient will simply modify the code and submit it as their own. Consider instead using a screen-sharing collaboration app, such as `http://webex.gatech.edu/`, to help someone with debugging if you're not in the same room.

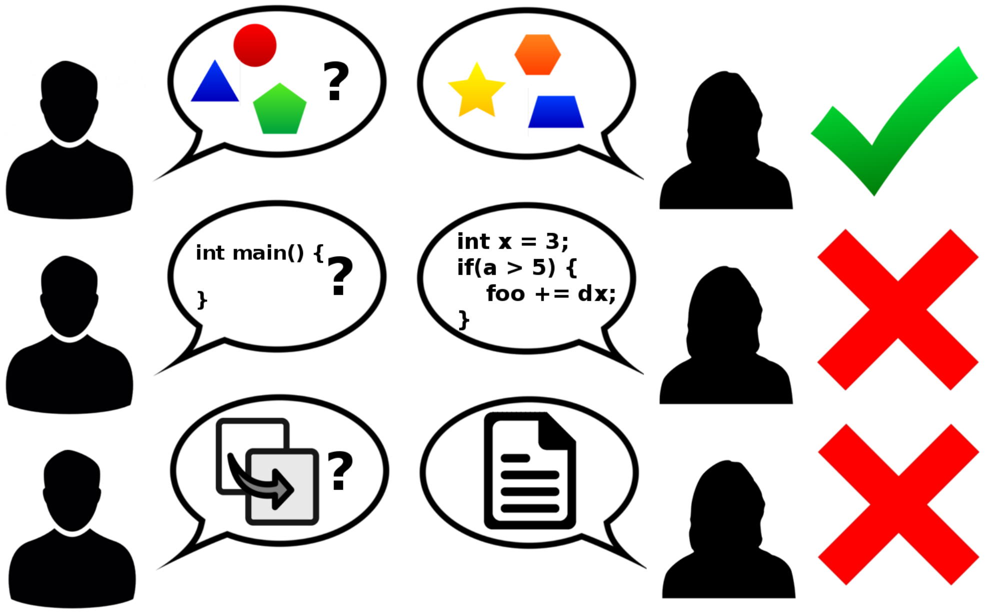

Figure 4: Collaboration rules, explained colorfully