CS 2110 Homework 3
Sequential Logic & State Machines

Shawn Wahi, Saahir Dhanani, Justin Hinckley, Nimish Mishra, Alice Zeng, John Ever

Fall 2021

Contents

1 Overview

1.1 Purpose

The purpose of this assignment is to practice implementing sequential logic circuits – from an RS Latch up to a Finite State Machine, using Circuitsim. You will build a register from the ground up. Then you will build a One Hot State Machine circuit, based on a provided state machine diagram (see Part 2 below). Then you will build a reduced state machine, using a K-Map to simplify your logic.

Objectives:

1. Understand how a register circuit works
2. To learn how to make a state machine
3. To become familiar with different state machine styles
4. To understand K-maps and simplification

1.2 Task

There are four parts to this assignment.

1. First, you will build a register in CircuitSim from the ground up.
2. Second, you will implement a one hot state machine Circuit in CircuitSim, based on the state transition diagram found below.
3. Third, you will complete a K-Map to simplify your state machine circuit, and implement the reduced state machine.
4. Fourth, you will build the program counter (PC) register.

Please read the rest of this document for more detailed instructions and hints.

1.3 Criteria

Your grade is based on: 1) the correct outputs from your circuits; and 2) not using any banned components. For part 3 (reduced state machine), you will lose points if your circuit does not correspond to your K-Map or if your circuit is not minimal. The grade you see on Gradescope may not be the final grade you receive, as we may run additional tests on your submission.

You will submit four files to Gradescope: `latches.sim`; `fsm.sim`; `kmap.xlsx`; and `PC.sim`.

You may submit your code to Gradescope as many times as you like until the deadline. We will grade your last submission. We have also provided a local checker that you can test your code with. Please submit your code to Gradescope at least once prior to the deadline, to ensure you are not encountering any issues submitting at the last minute.

2 Instructions

Part 1: For this part of the assignment you will build your own register from the ground up.

- Implement your circuits in the "`latches.sim`" file

Part 2: Given a simple state diagram, you will build a state machine in CircuitSim using the "one-hot" style of building state machines.

- The circuit will be implemented in the "One Hot FSM" subcircuit of the "`fsm.sim`" file

Part 3: Given the same state diagram from part 1, you will be minimizing the logic by using K-Maps.

- Fill out the K-Maps located in the spreadsheet named "`kmap.xlsx`"
- The reduced circuit will be implemented in the "Reduced FSM" subcircuit of the "`fsm.sim`" file

Part 4: Create the PC component, which is a 16 bit register.

- Create the PC (Program Counter), a register that satisfies the requirements specified in section 2.4 of this PDF
- The PC component will be implemented in the "`pc.sim`" file

Do not change/delete any of the input/output pins.

For parts 2 and 3 of this homework, you must use exactly 1 register. These are found under the Memory tab in CircuitSim. Failure to do so may result in large point deductions.

2.1 Part 1

For this part of the assignment you will build your own register from the ground up. For more information about each subcircuit refer to your textbook.

2.1.1 RS Latch

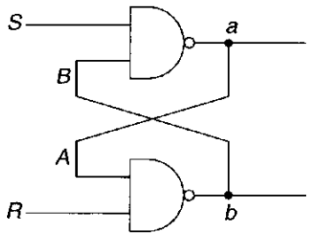
You will start by building a RS latch using NAND gates, as described in your textbook. The RS Latch is the basic circuit for sequential logic. It stores one bit of information, and it has 3 important states:

1. R=1 S=1 : This is called the **Quiescent State**. In this state the latch is storing a value, and nothing is trying to change that value.
2. R=1 S=0 : By changing momentarily from the Quiescent State to this state, the value of the latch is changed so that it now stores a 1.
3. R=0 S=1 : By changing momentarily from the Quiescent State to this state, the value of the latch is changed so that it now stores a 0.

Once you set the bit you wish to store, change back to the quiescent state to keep that value stored.

Notice that the circuit has two output pins; one is the bit the latch is currently storing, and the other is the opposite of that bit.

Note: In order for the RS Latch to work properly, you must not set both R and S to 0 at the same time.



- Build your circuit in the "RS Latch" subcircuit in the "`latches.sim`" file

2.1.2 Gated D Latch

Using your RS latch subcircuit, implement a Gated D Latch as described on the textbook.

The Gated D Latch is made up of an RS Latch as well as two additional gates that serve as a control. With that addition not only can we control what value is stored by the latch, but also when that value will be saved.

The value of the output can only be changed when Write Enable is set to 1. Notice that the Gated D Latch subcircuit only has one output pin, so you should disregard the inverse output of your RS Latch.

- Implement this circuit in the "Gated D Latch" subcircuit in the "`latches.sim`" file
- You are not allowed to use the built-in SR Flip-Flop in CircuitSim to build this circuit

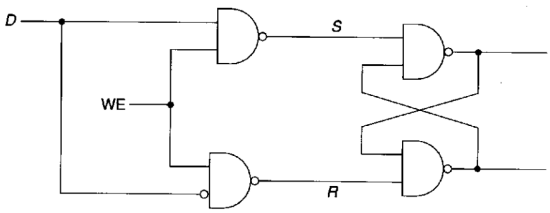


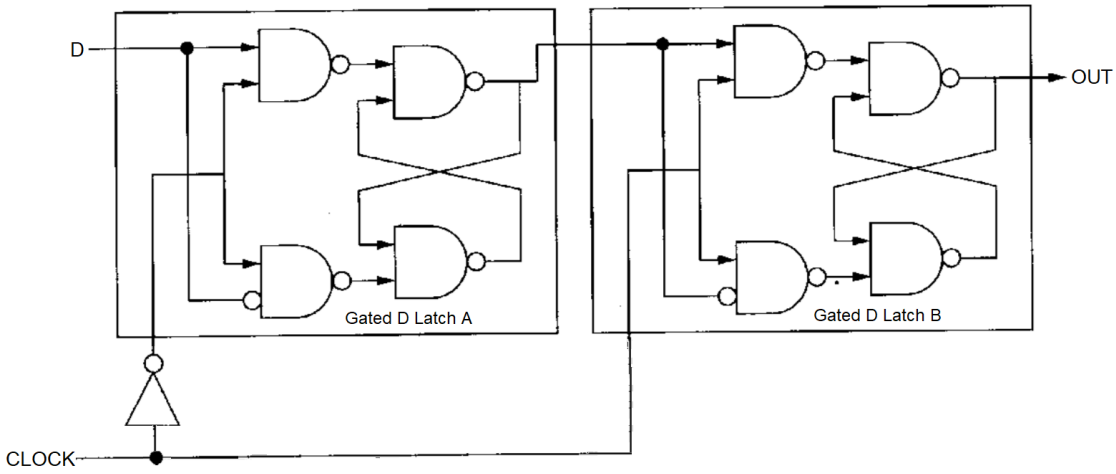
Figure 3.19 A gated D latch

2.1.3 D Flip-Flop

Using the Gated D Latch circuit you built, create a D flip-flop.

A leader-follower D flip-flop is composed of two Gated D latches back to back, and it implements edge triggered logic.

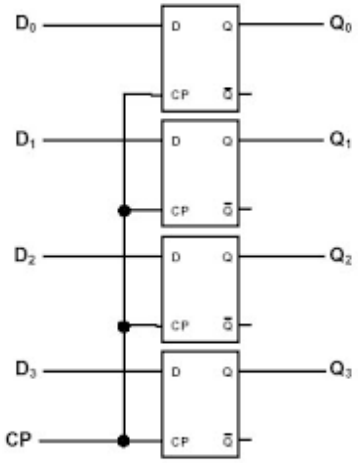
Your D flip-flop output should be able to change on the **rising edge**, which means that the state of the D Flip-Flop should only be able to change at the exact instant the clock goes from 0 to 1.



- Implement this circuit in the "D Flip-Flop" subcircuit in the "`latches.sim`" file.

2.1.4 Register

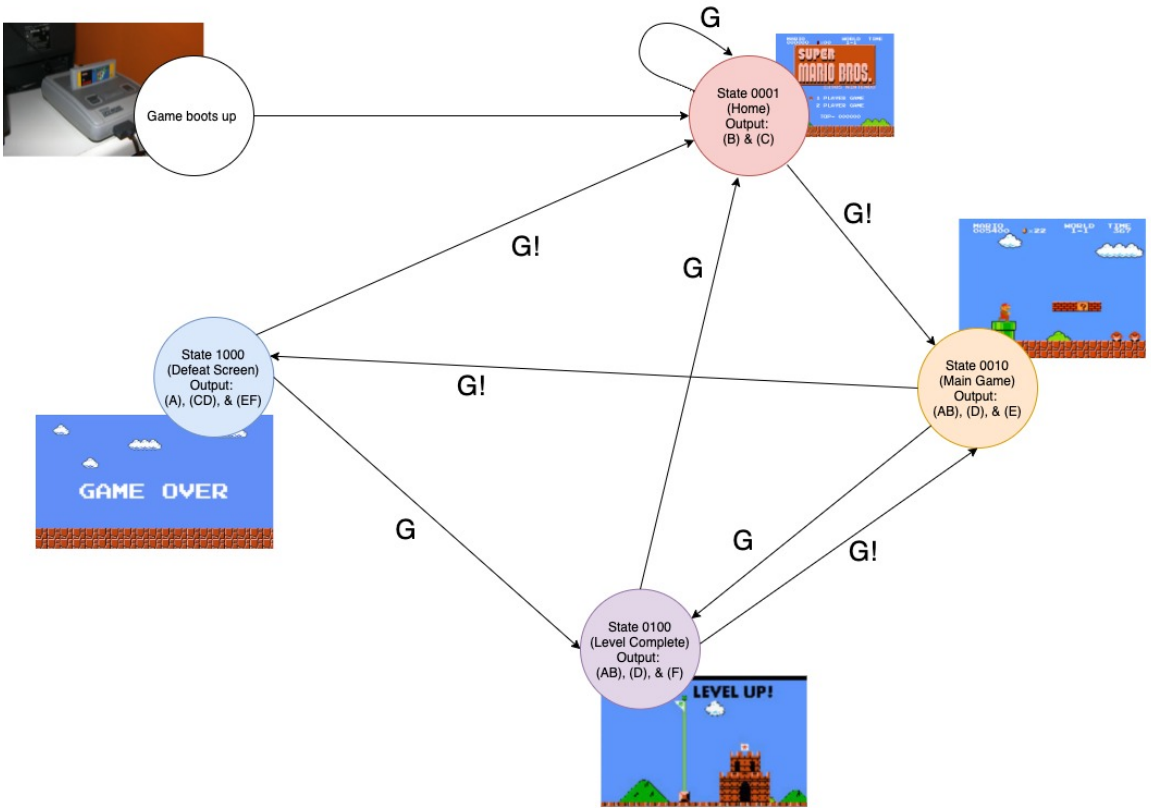
Using the D Flip-Flop you just created, build a 4-bit Register. Your register should also use edge-triggered logic. The value of the register should change on the rising edge.



- This circuit will be implemented in the "Register" subcircuit in the "`latches.sim`" file

2.2 Part 2

For this part of the assignment you are given the state machine transition diagram of a new video game that is currently in the early stages of production. The game has a start state, game state, victory state, and defeat state. Our game manager has one input button which is either pressed (G) or not pressed (G!). This computer has six outputs (A, B, C, D, E, F) that affect the screen, music, and other details in the game plays for the user.



Output	Screen
A	Defeat Screen
B	Select Screen
AB	Game Screen

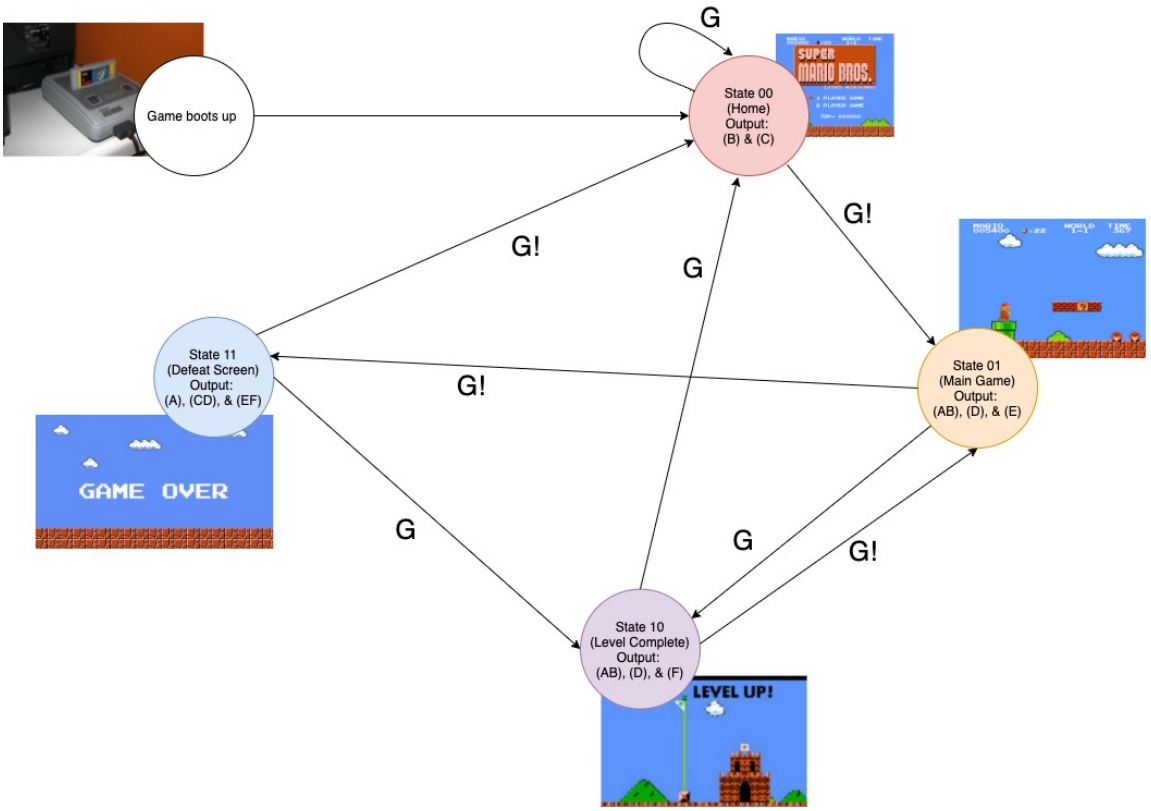
Output	Music
C	Intro Song
D	Main Game Music
CD	Defeat Music

Output	Other Details
E	Displaying Score
F	Confetti
EF	Leader Board

- You will be implementing this state transition diagram as a circuit using the one-hot style. Remember for one-hot you will have a register with the number of bits being the number of states of the FSM. Each bit corresponds to a state, and you are in that state if the corresponding bit is a 1. Since you can only be in one state at a time, exactly one of the bits can be 1 at each time (except when the machine starts up, when all the bits will be 0).
- You must implement this using one-hot. A template file `fsm.sim` has been given to you. Implement the state machine in the provided "one-hot state machine" subcircuit.
- Note that there are 3 inputs to the "One Hot FSM" subcircuit: CLK, G, and RST. The CLK input turning off and on repeatedly will be used to represent clock ticks in your circuit. The G input corresponds to whether or not G is on, as in the diagram above. RST resets the circuit (clears all the bits of your state register).

2.3 Part 3

Take a look at this state machine transition diagram:



- First, produce the K-maps for the state diagram above on the provided spreadsheet named "kmap.xlsx". Use the K-maps to produce the reduced Boolean expressions for the state machine.

  The inputs for each K-map are:

  - $S0$ = Current state least significant bit
  - $S1$ = Current state most significant bit
  - G = Input button

  The outputs make K-maps for are:

  - $N0$ = Next State least significant bit
  - $N1$ = Next State most significant bit
  - A, B, C, D, E, F

  Please Note: This State Machine is a Moore State Machine, meaning that the output values are determined solely by the current state (you should not use the N_1 and N_0 outputs or the G input for determining the values for A, B, C, D, E, F.

  - You will fill out one K-map per output and one per next state bit for a total of 8 K-maps (A, B, C, D, E, F, $N0$, $N1$). The respective K-maps are located in the `kmap.xlsx` file.
  - Your K-map must give the best solution groupings possible to receive full credit. This means you must select the optimal values for any don't cares (if applicable) in your K-maps to do this.

- It may be helpful to check with others on Piazza to see if your circuit is optimal. In order to do this without giving away your answer you may share the number of AND and OR gates used.
- **IMPORTANT:** The K-maps will be autograded. Because of this, there are a set of restrictions to how you must fill your K-maps to ensure you get full credit:
  * When you fill the row and column headers for your K-maps, you may only use the following variable names: $S0$, $S1$, and G. To negate a variable, you must use an apostrophe. Two adjacent variables with nothing in between are interpreted as an AND.
    Example label: $S0'G$
  * When writing the Boolean expressions resulted from your K-map groupings, you must use the same rules as the previous bullet point, but also use "`+`" for OR.
    Example grouping: For the Boolean expression `(NOT S0) OR (S1 AND G)`, write $S0'+S1G$
  * When filling in the cells of your K-map table, you must use `0`, `1`, and `x`.

- Implement this circuit in the "Reduced FSM" subcircuit of the provided `fsm.sim` file. You will lose points if your circuit does not correspond to your K-map or if your circuit is not minimal. You should use only the minimal components possible to implement the state machine.
- **HINT:** We recommend you make a truth table for the state machine to help organize the logic, and then transfer your answers to the K-maps. We've provided `truthtable.xlsx` to help with this.

  Note: You are not required to complete `truthtable.xlsx` or submit it anywhere; it is only provided for convenience when making your K-maps.

2.4 Part 4

You will need to build the PC (Program Counter) subcircuit. The PC register is a core component of computers to aid in program sequences.

The PC is a 16 bit register that holds the address of the next instruction to be executed. The contents of the PC must be updated depending on which MUX selectors and input signals are selected. There are three scenarios for updating the PC:

1. The contents of the PC are incremented by 1.
   Selected when PCMUX = 0b00
2. The result of the ADDR is the address of the next instruction. The output from the ADDR should be stored in the PC.
   Selected when PCMUX = 0b01
3. The value on the BUS is the address of the next instruction. The value on the BUS should be stored into the PC.
   Selected when PCMUX = 0b10

The PC should only be loaded on a rising clock edge when the proper input signal is on. Ensure that you don't reach the unused case (PCMUX = 0b11) of the circuit, or else spooky stuff might happen (undefined behavior). All the proper inputs have been provided (and the reset has already been connected).

3 Testing

To test your **circuits** locally, navigate to the directory with the latches.sim and fsm.sim files and run the tester JAR file with

```
java -jar hw03-tester.jar
```

Make sure to run the local autograder in a terminal in Docker.

To test your K-maps, please submit your `kmap.xlsx` to the "Homework 3: K-maps" assignment on Gradescope.

4 Deliverables

Submit the following files to the "Homework 3: State Machines" assignment on Gradescope:

- `latches.sim`
- `fsm.sim`
- `PC.sim`

Additionally, submit the following files to the "Homework 3: K-maps" assignment on Gradescope:

- `kmap.xlsx`

Note: The autograder may not reflect your final grade on the assignment. We reserve the right to run additional tests during grading.

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.

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.

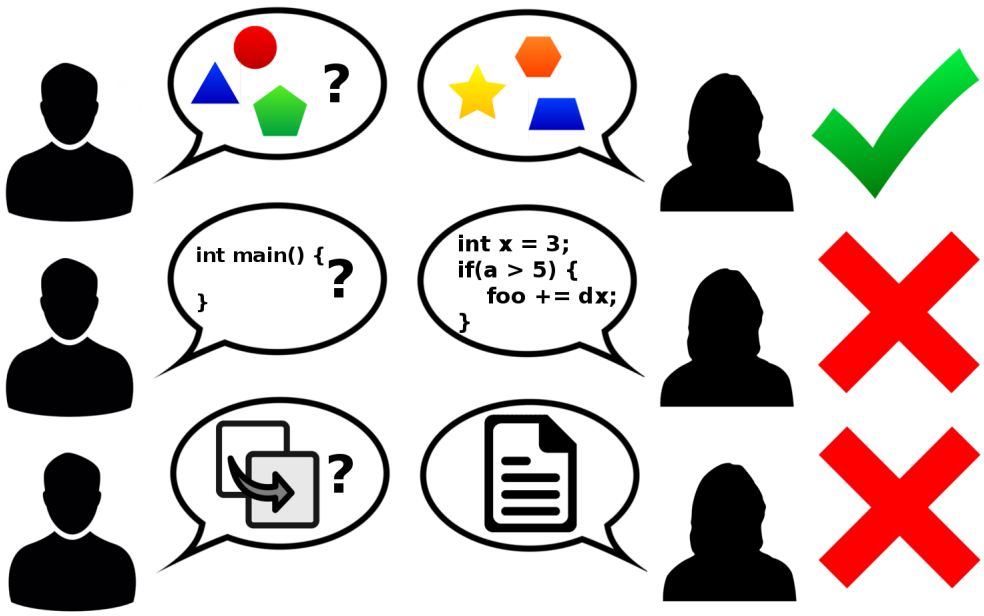


Figure 1: Collaboration rules, explained colorfully