CS 2110 Homework 04: Finite State Machines

Yuuna Hoshi, Michael Xu, and Vivian Thiebaut

Summer 2018

Contents

1 Overview

For this assignment, we'll be working with state machines and K-maps. This will be a 2 part assignment.

Objectives:

1. To learn how to make a state machine using basic logic gates
2. To become familiar with different state machine styles
3. To understand K-maps and their usefulness

2 Instructions

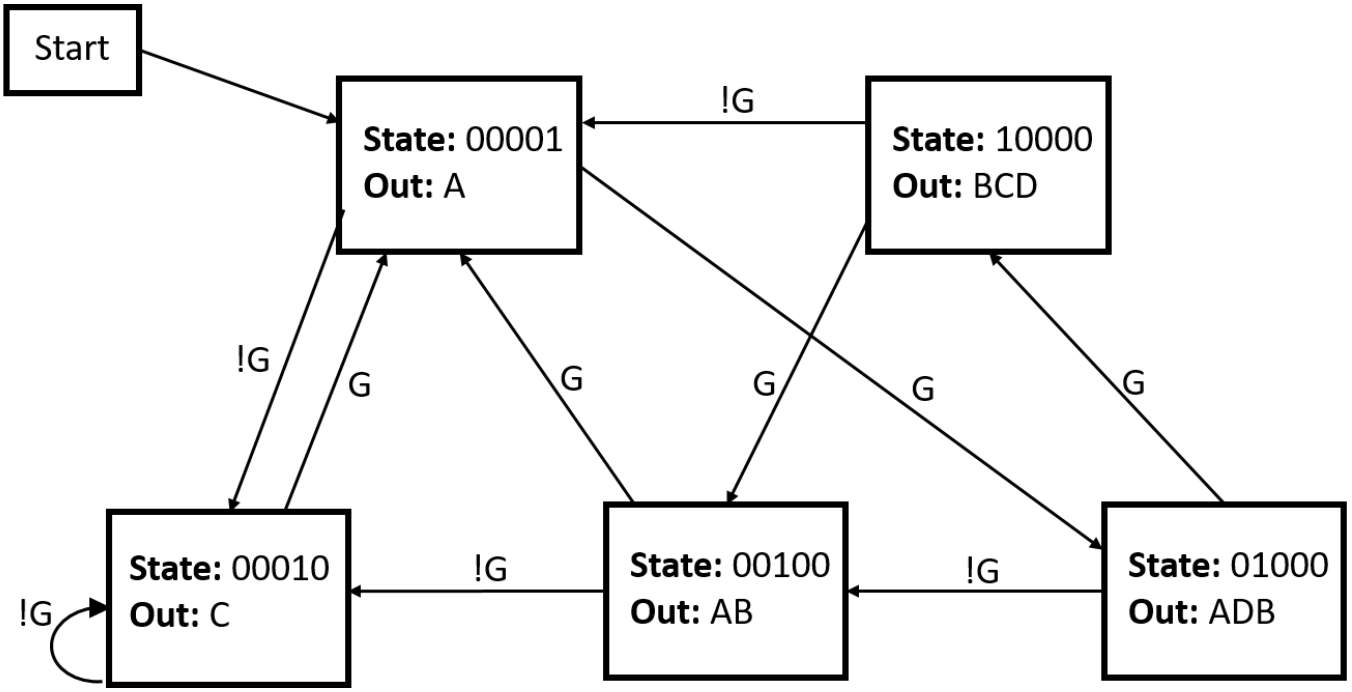
Part 1: Given a simple state diagram, you will build a state machine in CircuitSim using the "one-hot" style of building state machines.

Part 2: Given the same state diagram from part 1, you will build a reduced state machine. To do so, you'll need minimize the logic by using K-maps. After you have minimized the logic, implement the circuit in CircuitSim. Be sure to keep track of your K-maps! You will be turning them in along with your circuits.

For both parts 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

Take a look at this state machine transition diagram.



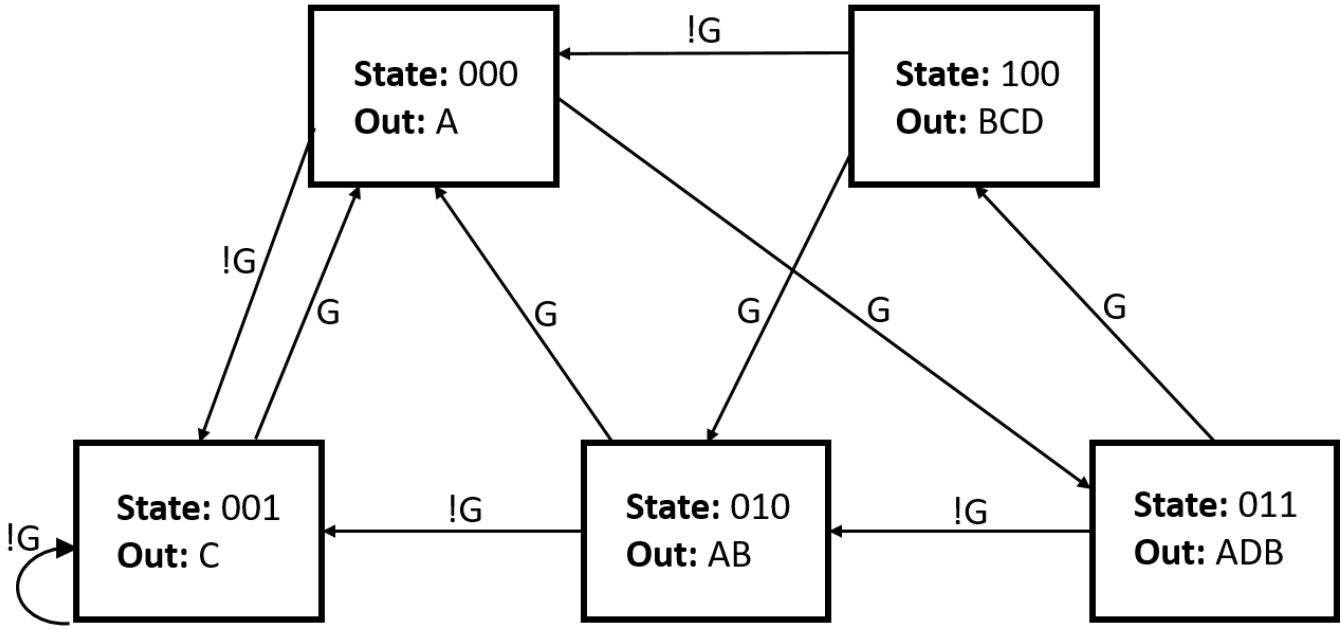
- You will be implementing this state transition diagram as a circuit using the one-hot style. Remember for one-hot you will have a register with the number of bits being the number of states you have. Each bit corresponds to a state, and you are in that state if the corresponding bit is a 1. Only one of these

bits will be on at a time (the only exception being when the state machine starts up). At most, one of the inputs will be turned on.

- You will need to implement a reset input for this circuit, which will return your state machine to its Start state. You may design this so that the reset happens either immediately when the reset button is pressed (asynchronously) or when it is on during the rising edge of the next clock cycle.
- You must implement this using one-hot. A template file hw04.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 state machine" 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 corresponds to an attempt to reset your circuit.

2.2 Part 2

Take a look at the following state diagram.



You will be implementing this state transition diagram as a reduced state machine, and **not in one-hot style**. You will be required to make a K-map and minimize the logic here.

- First, convert this state diagram to a truth table.
- Next, produce a K-map from the truth table. You will need one K-map per output and one per next state bit for a total of 7 K-maps (n2, n1, n0, A, B, C, D). Please be clear in labeling your K-maps. In the above diagram, the binary numbers on each state represent the state number. You must use these numbers in your K-map. For instance, if you see the number 011 then S2 = 0, S1 = 1, S0 = 1. This means that S2 is the most significant bit. Note that the states 110 and 111 aren't shown above. These are invalid states, and we don't care about the behavior of these states or which state they go to next. Use this when making your K-map. In your K-map, you must circle the groups (or highlight the groups) you have elected to use in your minimized SUM OF PRODUCTS expression. Your K-map

must give the BEST minimization possible to receive full credit. This means you must select the BEST values for the don't cares in your K-maps to do this.

- You must implement a reset input for this part of the assignment, which should reset the machine to State 000. As with your one-hot circuit, you may do this asynchronously or in time with the clock.
- Implement this circuit in the "reduced state machine" subcircuit of the provided hw04.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.
- 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. For example in the K-map below if you thought the answer was BD + ABC then you might ask if someone got better than 2 + 3. The 2 represents the number of factors (B and D) in the first term (BD), and the 3 represents the number of factors (A, B, C) in the second term (ABC).
- The outputs in your K-map should depend only upon the current state you are in. That is if the state is 010 and G is pressed, then A and B should be on. If the current state is 010 and G is not pressed, then A and B should still be on. The output does not depend on the G input, but only what state you are in.
- Your K-map must also choose the BEST values for the don't cares to achieve the minimal logic necessary to implement the state diagram as a circuit. For instance consider the following K-map (that is unrelated to the state machine we gave you):

AB\CD	00	01	11	10
00	0	0	0	0
01	0	1	X	0
11	0	X	1	X
10	0	0	0	0

Doing this the naïve way, you would select the two 1's in this K-map and be done, however this is not minimal. The X's in this K-map are don't cares, meaning you don't care if it is a 1 or a 0. You can use these don't cares to make your expression even more minimal. For this K-map you can let the X's in the center be 1's this way you can select a 2x2 square in the center. Also note the other don't care we will let be 0 as you do not get a better minimization by having it be a 1. You must do your minimization this way or else your K-maps will be wrong and you will lose points.

3 Testing Your Work

Flip through your circuit manually. Make sure you check all transitions and your circuit matches what is expected in the state diagram.

For a final check, we have also provided an autograder. To use the autograder, place `grader-summer18-tests-1.5.jar` in the same directory as your `hw04.sim`, and run the following:

```
java -jar grader-summer18-tests-1.5.jar
```

4 Rubric

Failure to follow these will result in a heavy point deduction, if not a zero, for this part.

1. Outputs should ONLY depend upon the current state.
2. Your K-maps should show the labeled groups and the minimized boolean expression as a sum of products. Do not factor anything else out. Your expression should not include any parenthesis, and the circuit shouldn't use XOR gates as part of the logic.
3. Your K-maps should give the BEST minimization. You should choose the BEST values for all of the don't cares.
4. States 111, 110 and 101 are not used so we DON'T CARE what is outputted or what the next state is.
5. Your K-map should match your circuit.
6. The left most bit of the state is S2, the right most is S0.
7. You are to do your truth table and K-maps on paper by filling out the provided worksheet. **hw4_kmaps.pdf**
8. Make sure your name is clearly visible in all of your files (including K-maps).
9. **You must use EXACTLY ONE REGISTER per state machine. No more, no fewer. This register MUST be named "reg" for our autograder to work.**

5 Deliverables

1. hw04.sim
2. hw04_kmaps.pdf

Submit both files on **Canvas**. You do not need to submit your truth table.

For your K-maps, please scan/take a picture of your work, and make sure it is legible.

We will also demo this assignment. More information will be provided at a later date.

Part 1:

1. The modified "one-hot state machine" subcircuit in the hw04.sim file, which implements your one-hot state machine.

Part 2:

1. Your K-maps, and reduced boolean expressions in a scanned version of hw4_kmaps.pdf.
2. The modified "reduced state machine" subcircuit in the hw04.sim file, which implements the minimized state machine from your K-Maps.

Download and test your submission to make sure you submitted the right files.

6 Rules and Regulations

6.1 General Rules

1. Starting with the assembly homeworks, Any code you write (if any) must be clearly commented and the comments must be meaningful. You should comment your code in terms of the algorithm you are implementing we all know what the 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.

6.2 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.

6.3 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. 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.

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, 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.

6.4 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

6.5 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, as well as help each other debug code. What you shouldn't be doing, however, is paired programming where you collaborate with each other on a low level. 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, and it is often the case that the recipient will simply modify the code and submit it as their own.

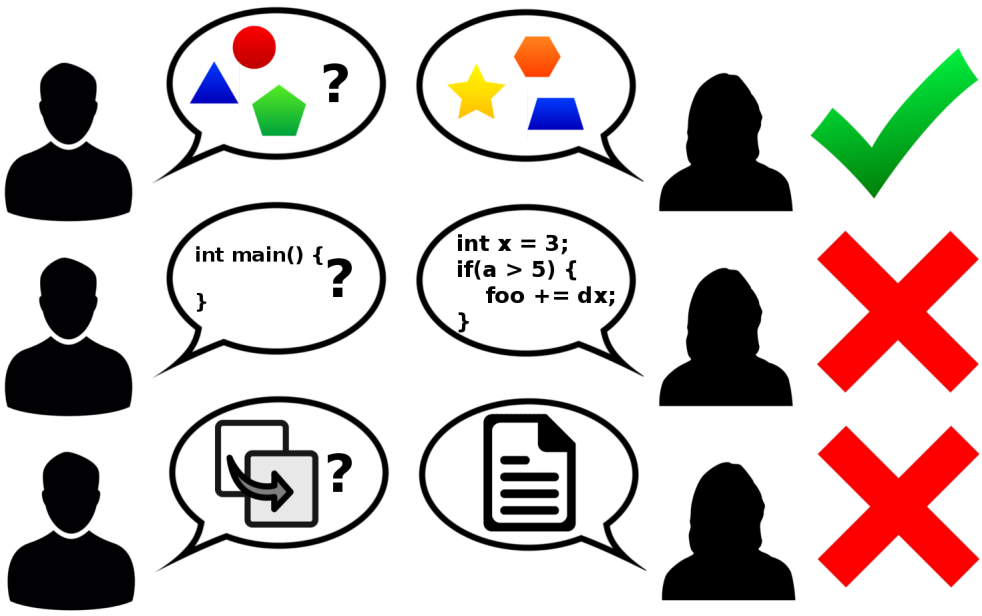


Figure 1: Collaboration rules, explained