

CSE 140 Homework Three

August 28, 2014

*Only **Problem Set Part B** will be graded. Turn in only **Problem Set Part B** which will be due on September 4, 2014 (Thursday) at 4:30pm.*

1 Problem Set Part A

- Roth&Kinney, 6th Ed: 9.14
- Roth&Kinney, 6th Ed: 9.18
- Roth&Kinney, 6th Ed: 9.19
- Roth&Kinney, 6th Ed: 9.27
- Roth&Kinney, 6th Ed: 9.29
- Roth&Kinney, 6th Ed: 9.31
- Roth&Kinney, 6th Ed: 9.36
- Roth&Kinney, 6th Ed: 11.2
- Roth&Kinney, 6th Ed: 11.15
- Roth&Kinney, 6th Ed: 11.16
- Roth&Kinney, 6th Ed: 14.4
- Roth&Kinney, 6th Ed: 14.12
- Roth&Kinney, 6th Ed: 14.13
- Roth&Kinney, 6th Ed: 14.17
- Roth&Kinney, 6th Ed: 14.23

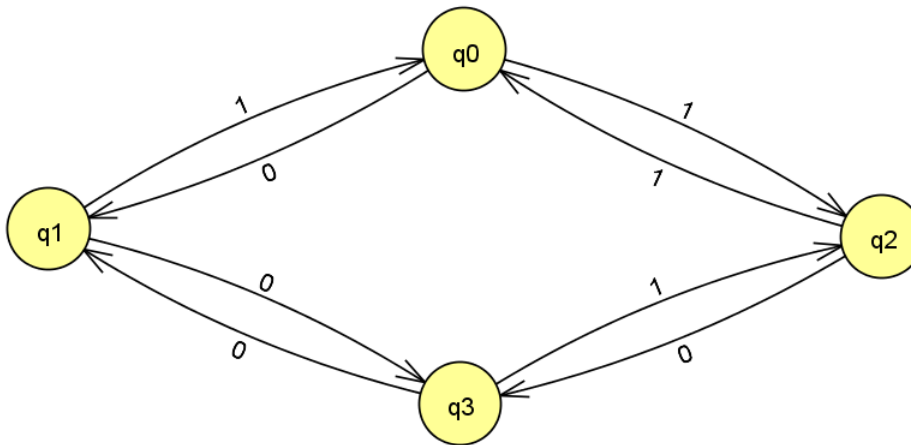
- Roth&Kinney, 6th Ed: 15.3
- Roth&Kinney, 6th Ed: 15.4
- Roth&Kinney, 6th Ed: 15.10
- Roth&Kinney, 6th Ed: 15.11
- Roth&Kinney, 6th Ed: 15.15
- Roth&Kinney, 6th Ed: 15.22
- Roth&Kinney, 6th Ed: 15.23
- Roth&Kinney, 6th Ed: 15.27
- Roth&Kinney, 6th Ed: 15.28
- Roth&Kinney, 6th Ed: 15.33

2 Problem Set Part B

1 (State Minimization)

When doing state minimization, one of the main factors in determining if two states are equivalent is if they produce the same output on the same input. The following question thus explores the impact that output specifications have on state minimization.

(Part A) As you can tell, the FSM shown below has 4 states. However, if we assume that this is a Mealy Machine, state minimization may or may not be possible depending on what the outputs are. We ask you to explore this phenomenon by specifying output combinations in the 4 tables below, in which the first table will result in an FSM description that cannot be minimized, the second table will result in an FSM that can be minimized to 3 states, the third table will result in an FSM that can be minimized to 2 states, and the fourth table will result in an FSM that can be minimized to a single state.



<i>States\Input</i>	0	1
<i>q0</i>	<i>q1</i> /—	<i>q2</i> /—
<i>q1</i>	<i>q3</i> /—	<i>q0</i> /—
<i>q2</i>	<i>q3</i> /—	<i>q0</i> /—
<i>q3</i>	<i>q1</i> /—	<i>q2</i> /—

(a) Not Minimizable

<i>States\Input</i>	0	1
<i>q0</i>	<i>q1</i> /—	<i>q2</i> /—
<i>q1</i>	<i>q3</i> /—	<i>q0</i> /—
<i>q2</i>	<i>q3</i> /—	<i>q0</i> /—
<i>q3</i>	<i>q1</i> /—	<i>q2</i> /—

(b) Minimizable to 3 states

<i>States\Input</i>	0	1
<i>q0</i>	<i>q1</i> /—	<i>q2</i> /—
<i>q1</i>	<i>q3</i> /—	<i>q0</i> /—
<i>q2</i>	<i>q3</i> /—	<i>q0</i> /—
<i>q3</i>	<i>q1</i> /—	<i>q2</i> /—

(c) Minimizable to 2 states

<i>States\Input</i>	0	1
<i>q0</i>	<i>q1</i> /—	<i>q2</i> /—
<i>q1</i>	<i>q3</i> /—	<i>q0</i> /—
<i>q2</i>	<i>q3</i> /—	<i>q0</i> /—
<i>q3</i>	<i>q1</i> /—	<i>q2</i> /—

(d) Minimizable to 1 state

(Part B) We will now consider an extension of the output space to consider *don't-cares*. As you might have figured out, this is a non-trivial extension that results in 2 potential pitfalls. The first pitfall is that an unspecified output can assume one of several possible values, which results in an exponential increase in computational complexity as you contemplate which combination of the unspecified output values will be most beneficial in aggressively minimizing the FSM. A second pitfall is that transitivity in equivalence groups may not necessarily hold; if an unspecified output is assumed to hold one value to satisfy an equivalence, any equivalence that requires that output to hold a different value will thus necessarily be considered invalid, yet either of these pairs may individually denote a valid equivalence while their transitive union may not hold.

Keeping these pitfalls in mind, we ask that you perform a state minimization on the following set of next-states and outputs. While you may be tempted to draw several implication tables and choose the one that results in the fewest number of states, we urge you to consider approaching the problem at a higher level in order to achieve the desired minimization.

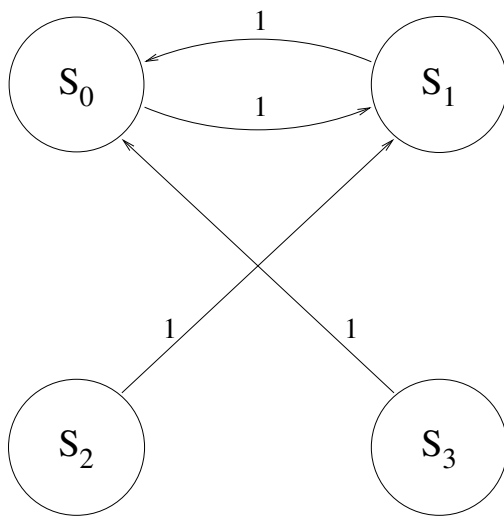
Please draw the resulting state partitions, and show how you arrived at them and the reasoning you used that led you to believe that this is the optimally minimized FSM.

<i>States</i> \ <i>Input</i>	0	1
S_0	$S_1/0$	S_0/X
S_1	S_2/X	$S_1/1$
S_2	$S_1/1$	$S_0/1$
S_3	$S_5/1$	S_4/X
S_4	$S_5/0$	$S_4/1$
S_5	S_3/X	$S_5/1$

2 (State Encoding)

As you have learned in lecture, several methodologies may be employed when one does state encoding. Perhaps the most interesting of the bunch are the *prioritized adjacency* heuristics, which facilitate the circling of larger cubes when implementing your next-state logic using **D** flip-flops. As a refresher, the first prioritized adjacency heuristic involves facilitating cube expansion in the initial-state space (i.e. the Q_1Q_0 space in the Karnaugh map below), whereas the second heuristic manifests larger cubes across the input space (i.e. the I space in the Karnaugh map below). One drawback to these heuristics is that, individually, they only help form cubes of 2 terms, so in this question we will expand on these state encoding methodologies in order to generate cubes of 4 terms.

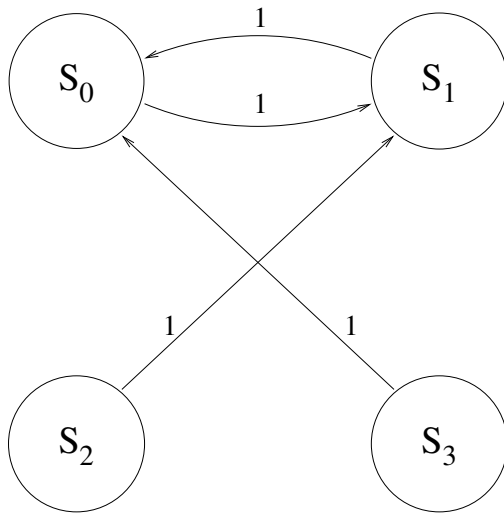
The following 2 parts of this question require you to complete the partially specified 4-state FSM below by both drawing the remaining arrows as well as deducing the state encodings that enable the circling of 4-term cubes for both flip flops. You are also asked to fill in the next-state table so that you can verify that you end up indeed generating cubes of 4. Of course, you realize that if your Karnaugh maps for the next state logic for these two flip flops are composed of nothing but 4 (min/max)term cubes, you will end up with no gates whatsoever for the next-state logic, to be composed of nothing but input or flip-flop output lines driving the D-inputs of the flip flops.



$I \backslash Q_1Q_0$	00	01	11	10
0				
1				

(Part A) As you can probably tell, a textbook application of the first prioritized adjacency heuristic on the partially specified FSM shown above dictates that the leftmost 2 states must be Hamming adjacent and that the rightmost 2 states must be Hamming adjacent as well. However, two sets of state encodings may occur, one that yields a cube of 4 for one of the flip flops, and another that only yields a cube of 2. You can implement this next-state logic in either sum-of-products or product-of-sums form, so your cubes of 4 can be comprised of minterms and/or maxterms. Keeping this in mind, please come up with a state encoding that both satisfies the first prioritized adjacency heuristic for the partially specified FSM above as well as generating a cube of 4 for one of the flip flops. Fill in the portions of the Karnaugh Map above that you are able to, and write the state encodings that correspond to each of the four states, S_0, S_1, S_2 , and S_3 .

(Part B) Now that you have generated the state encodings, we ask that you draw the remaining arrows that designate the 0-input transition for each state such that you can generate a cube of 4 for the remaining flip flop. As you can probably tell, one such possible arrow designation results in both the 1 and 0 inputs leading to the identical next-state for every state, but this results in an FSM with a fair amount of redundancy wherein the inputs are irrelevant (and thus should have been eliminated). Thus, we ask that you draw the arrows for a case that doesn't manifest this property but does result in a cube of 4 for the second flip flop. Please fill in thereupon the remaining boxes of the Karnaugh Map that facilitate the formation of 4 minterm cubes. Your final next-state table should result in no gates whatsoever in the next-state logic but only connections to the input and next-state outputs. For your convenience, the partially specified FSM and the Karnaugh map from the previous page are shown below. Please specify which of the two diagrams contains your final answer.



$I \backslash Q_1 Q_0$	00	01	11	10
0				
1				

3 (Decoders, Shifters, and Priority Encoders)

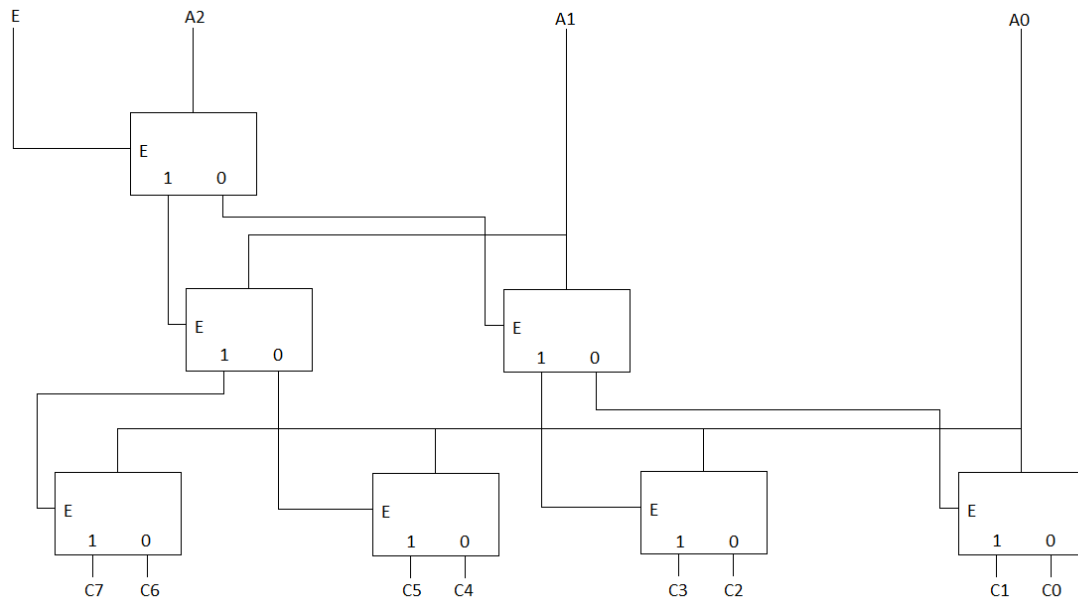
You know the type. You may have even had some friends like that. The type who goes left when everybody goes right. The roommate who insists on sleeping when everybody else is awake and who plays his music at full volume when you are trying to catch a snooze. The driver who drives the wrong way on one-way streets (but those are harder to run into, or perhaps easier on the highway, as natural selection seems to be quite effective at sieving them out). But leaving the driver aside, in general they are harmless, eccentric folk who half the time come up with something quite interesting, creative, and useful, and the other half of the time they fail miserably. The trick is to determine when they are on the money and when they are way off, but that is a story that lends itself to a volume of its own.

Just as you graduated from college, and you thought that finally you were through with dealing with such eccentric folks (colleges seem to have a way of breeding them up), you hear rumors in the company you have joined of this one eccentric engineer who seems to draw confirmation of his eccentricities in the duality theorem and seems to be going around proclaiming pithy aphorisms about duality and muttering something about Manichaeism to boot. In short, somebody to be avoided, particularly when you are just starting your career. The trouble is that the CEO is of a dual mind about this character and loves the time when his eccentricities prove right, making the company tons of money, and hates the havoc particularly the wrong ideas seem to be wreaking in the company.

As you are the youngest member of the company, the CEO has tasked you with the impossible task of shadowing this eccentric fellow. His dual goals in having you shadow him are to both ensure that you keep him away from the rest of the hard-working, God-fearing engineers in the company, but also to decipher his ideas and analyze them for correctness and viability and immediately alert the CEO if an idea proves to have the possibility of a breakthrough. As you start in this task (and are fast wondering how you could immediately apply for another job), you have run into 3 ideas that this eccentric fellow has come up with and are spending your few hours of the night trying to relax a bit from the physical exhaustions of the full-court press on him you need to do throughout the day while at the same time trying to do a technical analysis of these ideas. Here they are:

(Part A) The eccentric engineer has been suggesting that a 1-to-2 decoder functionality whose gate-level representation you easily developed in class can be just as well implemented with OR-gates instead of AND-gates. Please refresh your memory (and ours) by first writing the Boolean equations and/or schematic of the 1-to-2 decoder. Once you have your memory refreshed, do you think you still have a 1-to-2 decoder functionality of some sorts if you replace the AND-gates with OR-gates? In what sense? Please provide a truth table and describe the similarities and differences.

(Part B) The problem is that this eccentric fellow is not only changing the building blocks of the 1-to-2 decoder but furthermore insists on working on a 3-to-8 decoder along the lines of the 3-to-8 decoder block diagram that you saw in class that we have reproduced below. He seems to have something up his sleeve and seems rather hopeful that a decoder-like functionality can be thus extricated. You have been pondering for a while now what change(s), if any, may need to be made and how this can be construed to have a decoder-like functionality. Have you made any progress?



(Part C) The second area this fellow is obsessed about seems to be the selectors. You have seen him take furtive glances at the gate-level implementation of a 4-to-1 selector and flipping the AND to OR and the OR to AND gates. You wonder what he might be musing about and start off by refreshing your memory and writing the logic expression(s) for the output of a 4-to-1 selector. After you refresh your memory of the boolean expression(s) and/or logic schematic, you do the same flip this fellow seemed to be musing about and write down the resultant truth table. Is this a 4-to-1 selector functionality? What do you think are the similarities and differences?

(Part D) The other part he seems to be playing with is a 16-to-1 selector that is built from three 4-to-1 selectors. He seems to be focused on a conventional 16-to-1 selector implementation thus constructed and every once in a while seems to utter words such as “Eureka, it works!” You are perplexed as to whether the conventional 16-to-1 selector comprised of three 4-to-1 selectors can be made to be a selector if the constituent pieces are the flipped designs utilized in (Part C). To refresh your memory, first write clearly the 16-to-1 conventional selector thus constructed and then ponder the question of whether this can have a selector-like functionality if the underlying 4-to-1 selectors are the ones with the gates flipped. What are the similarities and differences?

(Part E) What really blew your top and convinced you that you probably do not have a rosy future in this company is when you saw him staring at priority encoders. Priority encoders as you remember from your studies in CSE 140 a short couple of years ago was one of the most perplexing topics of Chapter 5 of your book. Now you need to go back and refresh your memory regarding a 2-to-1 priority encoder. Once you have written the Boolean expressions for the 2-to-1 priority encoder you studied in class, you ponder how the flipped from gates of the conventional priority encoder can be construed to exhibit a priority encoder functionality by writing out the truth table of this and providing an explanation of its priority encoder-like functionality. Do so below.

(Part F) The next step of looking at the 8-to-3 priority encoder implemented out of 2-to-1 priority encoders and 2-to-1 selectors and hearing his mutterings yet once again about “Eureka” evokes images in your mind of somebody drowning in an Archimedean bath (and hopefully of the perpetrator of the crime never being identified). But the hours are running short, and you decide to give up on these pleasant dreams and instead to concentrate on the question of how the 8-to-3 priority encoder shown below needs to be modified to make it something that has the functionality that is expected. What happens if the constituent parts of priority encoders are of the flipped kind as in (Part E)? Should the 8 outputs be permuted? In what order? What kinds of selectors should be used, the regular 2-to-1 or the flipped ones he was working on before? What is the interpretation of the output bits? The night is young and you are flipping between unsolved bathtub murder mysteries and these technical questions, when you decide to call it a night and top off this test with your answer to this subpart.

