

CSE 140 Homework Three

August 29, 2013

*Only **Problem Set Part B** will be graded. Turn in only **Problem Set Part B** which will be due on September 5, 2013 (Thursday) at 4:00pm.*

1 Problem Set Part A

- Roth&Kinney, 6th Ed: 8.2
- Roth&Kinney, 6th Ed: 8.6 - 8.8
- Roth&Kinney, 6th Ed: 8.11
- Roth&Kinney, 6th Ed: 9.14
- Roth&Kinney, 6th Ed: 9.18 - 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: 13.3(a)(b)
- Roth&Kinney, 6th Ed: 13.4(a)
- Roth&Kinney, 6th Ed: 13.8(a0
- Roth&Kinney, 6th Ed: 13.10(a)
- Roth&Kinney, 6th Ed: 14.4
- Roth&Kinney, 6th Ed: 14.12 - 14.13
- Roth&Kinney, 6th Ed: 14.17
- Roth&Kinney, 6th Ed: 14.23
- Roth&Kinney, 6th Ed: 15.3 - 15.4
- Roth&Kinney, 6th Ed: 15.10 - 15.11
- Roth&Kinney, 6th Ed: 15.15
- Roth&Kinney, 6th Ed: 15.22 - 15.23
- Roth&Kinney, 6th Ed: 15.27 - 15.28
- Roth&Kinney, 6th Ed: 15.33

1 (Static Hazards) As you have learned in class, static hazards manifest themselves when adjacent minterms (or maxterms) are not covered by a common implicant (or implicate). For example, the function $x'y' + xz$ contains a static-1 hazard because minterms $x'y'z$ and $xy'z$ are not covered by a common implicant, as shown in the K-map below, where one of the prime implicants in a minimal cover is bolded and the other is italicized:

x/yz	00	01	11	10
0	<i>1</i>	1	0	0
1	0	1	1	0

(Part A) For the following functions, please specify if

- The function will generate a static-1 hazard
- There are other minimal representations of the same function that will also generate a static-1 hazard
- If there is another equivalent function that will also generate a static-1 hazard, report the function

$$xy' + y'z + x'y$$

The function will/will not (circle one) generate a static-1 hazard.

There are/aren't (circle one) other minimal representations of the same function that will generate a static-1 hazard.

One other static-1 hazard representation is: (put N/A if there are none)

$$xy' + wz + yz$$

The function will/will not (circle one) generate a static-1 hazard.

There are/aren't (circle one) other minimal representations of the same function that will generate a static-1 hazard.

One other static-1 hazard representation is: (put N/A if there are none)

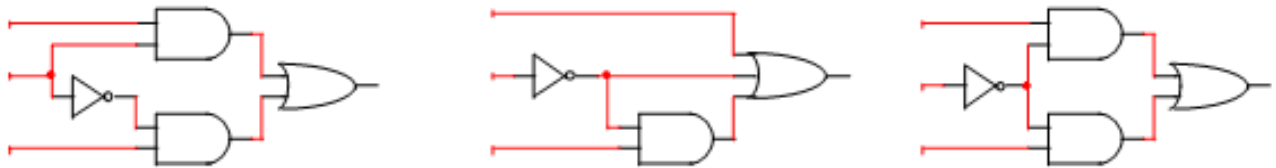
$$w'z + wz' + xy'z$$

The function will/will not (circle one) generate a static-1 hazard.

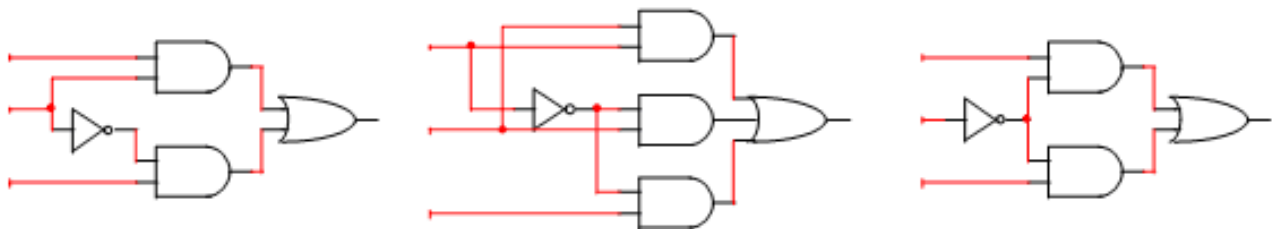
There are/aren't (circle one) other minimal representations of the same function that will generate a static-1 hazard.

One other static-1 hazard representation is: (put N/A if there are none)

(Part B) How many 3-variable 3-minterm functions contain static-1 hazards in their minimal implicant cover? Please explain your reasoning. If there are static-1 hazards, please circle the general schematic form in which such a hazard will take place.



(Part C) How many 3-variable 4-minterm functions contain static-1 hazards in their minimal implicant cover? Please explain your reasoning. If there are static-1 hazards, please circle the general schematic form in which such a hazard will take place.



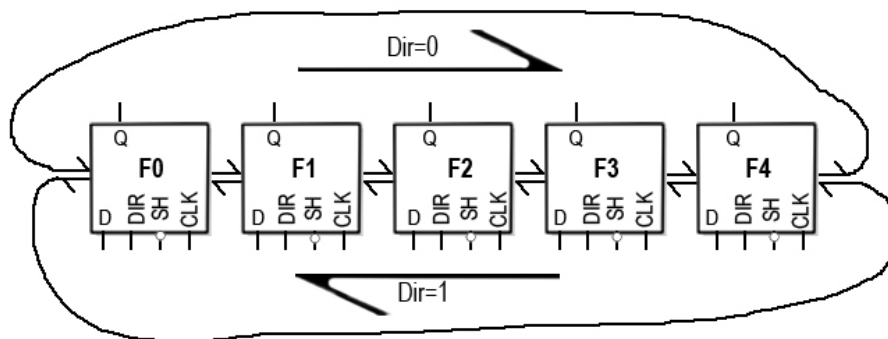
(Part D) How many 3-variable 4-minterm functions that contain static-1 hazards in their minimal implicant cover will also manifest static-0 hazards in their minimal *implicate* cover?

2 (Flip-Flops, Schlip-Schlops) After graduating this quarter as the top student in your CSE 140 class, you land a job at a famous but extremely secretive chip development company. Very early on, you detect that while this company is incredibly innovative, its managers and your fellow employees are just as incredibly strange. Things are done differently here. You feel uncertain that the skills you learned in your classes will apply here.

(Part A) To your delight, your first assignment seems simple and familiar. You are to design a modulo-5 counter that either counts up or down, depending on the inputs “C” and “D”, where “C” denotes whether the state should change on the next clock cycle, and “D” determines the direction, D=0 for counting up and D=1 for counting down. To make it even easier, you are to implement this FSM with a **hot-one encoding**, such that each of the five Flip-Flops, $F_0 \dots F_4$, outputs a 1 *only* when the counter is currently outputting the same value as its own index (when F_2 is on, for instance, the value in the counter is “2”). This resembles an assignment you had completed in one of the classes you took during your final quarter at UCSD, so you are quickly able to throw some **D-Flip-Flops** together to produce the correct FSM. Draw the state diagram for the FSM you design, explicitly labeling each state with the values in each of the Flip-Flops, $F_0 \dots F_4$. Describe the next-state logic for one of the Flip-Flops (a drawing is not necessary, but will suffice).

(Part B) Not long after sending your design specifications to the manufacturing department, your new boss stomps through your door looking very annoyed. She tells you that you have screwed up your first project, and that if you have any hope of keeping your job after botching such an elementary task, you had better explain the implications of the errors in your FSM.

She explains that if you had watched what you were doing, you would know that you implemented the FSM with Schlip-Schlops. Part of you wants to ask what the hell a Schlip-Schlop is, but most of you is afraid of being Schlapped in the face for it, so you maintain your paralysis while your boss explains that a Schlip-Schlop is like a Flip-Flop except that it has built-in shifting capabilities. It takes in an “SH” input (which is complemented), and a “DIR” input. On the rising edge of a clock cycle, the D-Schlip-Schlop behaves like a D-Flip-Flop, storing the value at its D pin. However, on the falling edge, the Schlip-Schlops (which are pre-connected to one another as shown below) collectively perform a shift (or not, if SH=0) in the direction specified by “DIR” (0 for right, 1 for left).



You remember seeing the “SH” and “DIR” inputs on the Schlip-Schlops you had used, but, being unfamiliar with them at the time, you simply grounded them. You quickly realize how this affects the behavior of your FSM, and you seize your chance to redeem yourself. Explain how your implementation actually behaves by redrawing your state diagram below.

(Part C) After you finish your explanation, which you delivered with an external confidence that in no way matched your internal state, your boss pauses for a moment. Then she laughs, and explains that she was only having a bit of fun. Everyone makes this mistake on their first day, that you were never really in any trouble, and that you should have seen your face. The Schlip-Schlop was a relatively new invention anyway, and she was just using the idea to test how you react in a world that sometimes seems to shift under your feet. She gives you a peppermint to ease your mind while you figure out how you are going to restore the FSM functionality you intended to make in (Part A). Explain how you would modify your inputs to streamline your next-state logic such that you would have to add as few logic gates as possible.

(Part D) After making your explanation, your boss asks you to ponder what your next-state logic would have to be if, using the same input configuration you came up with in (Part C), you were supplied with an array of T-Schlip-Schlops instead of D-Schlip-Schlops (T-Schlip-Schlops, of course, behave like T-Flip-Flops on the rising edge of the clock input, but for the shifting capability previously described).

3 (State Encoding) As you have learned in class, the first two rules of the prioritized adjacency state encoding heuristic attempt to cluster common values in order to enable the generation of larger cubes to simplify the next-state logic. The rules are reproduced below for your convenience:

1. If two states S_i and S_j both transition to state S_k on the same input a , then the encodings of S_i and S_j should be hamming adjacent.
2. If a state S_i transitions to state S_j on input a and to state S_k on input b , and a and b are hamming adjacent, then the encodings of S_j and S_k should be hamming adjacent.

The benefits of these rules are readily apparent should one apply them in a D flip-flop implementation, as the first rule induces commonalities in all bits in question of the next-state logic on the Karnaugh Map, while the second rule induces commonalities in all but one bit on the parts of the next-state logic affected by the rule on the Karnaugh Map. In this question, we ask that you consider the impact of these heuristics with respect to the other 3 flip-flop implementations: T, SR, and JK.

Since the two rules we discussed induce adjacent commonalities in the Karnaugh Map, we ask you to focus in your answers on the effect of flip-flop selection on the number of commonalities. As you know, a commonality is violated when the two adjacent bits differ, i.e. one is a “0” while the other is a “1”. We will stick to the same definition of commonality even in the face of “don’t-cares,” implying that the only time when a commonality is violated is when the adjacent bits have complementary values. Consequently, not only pairs such as “00” and “11” but also “0X”, “X0”, “X1”, “1X”, and “XX” should also be construed to display commonality.

Some of you may find it helpful to draw Karnaugh Maps displaying the induced commonalities in the case of D flip-flops and think about the impact of the various flip-flops on adjacent pairs displaying these commonalities.

(Part A) The first implementation to consider is that constructed with **T flip-flops**. Because T flip-flops have the same number of inputs as D flip-flops, at first glance one would expect the number of commonalities to be identical to that of the D flip-flop. However, further analysis might yield one of the cases enumerated below. **For each of the two aforementioned rules, please choose an option below and provide adequate justification** as to how the number of pairs with guaranteed commonalities in a T flip-flop implementation compares with that of a D flip-flop implementation. Your answers for the two rules may not be the same.

- I:** The number of commonalities induced by the rule is identical to what is present in a D flip-flop implementation.
- II:** The number of commonalities induced by the rule exceeds those in the D flip-flop implementation by 1.
- III:** The number of commonalities induced by the rule is 1 less than those in the D flip-flop implementation.
- IV:** The number of commonalities induced by the rule exceeds those in the D flip-flop implementation by a fixed, known quantity greater than 1 (but you don't need to report the exact quantity).
- V:** The number of commonalities induced by the rule is less than those in the D flip-flop implementation by a fixed, known quantity greater than 1 (but you don't need to report the exact quantity).
- VI:** The number of commonalities induced by the rule exceeds those in the D flip-flop implementation by a quantity that can only be determined on a case-by-case basis.
- VII:** The number of commonalities induced by the rule is less than those in the D flip-flop implementation by a quantity that can only be determined on a case-by-case basis.

(Part B) The next implementation to consider is that constructed with **SR flip-flops**. Because SR flip-flops double the number of inputs, one would expect that by using the 2 rules above, the commonalities in the bits of the next-state logic should double as well. However, further analysis might yield one of the cases enumerated below. **For each of the two aforementioned rules, please choose an option below and provide adequate justification** as to how the number of pairs with guaranteed commonalities in an SR flip-flop implementation compares with that of a D flip-flop implementation. Your answers for the two rules may not be the same.

- I:** The number of commonalities induced by the rule is exactly double what is present in a D flip-flop implementation.
- II:** The number of commonalities induced by the rule exceeds double those in the D flip-flop implementation by 1.
- III:** The number of commonalities induced by the rule is less than double those in the D flip-flop implementation by 1.
- IV:** The number of commonalities induced by the rule exceeds double those in the D flip-flop implementation by a fixed, known quantity greater than 1 (but you don't need to report the exact quantity).
- V:** The number of commonalities induced by the rule is less than double those in the D flip-flop implementation by a fixed, known quantity greater than 1 (but you don't need to report the exact quantity).
- VI:** The number of commonalities induced by the rule exceeds double those in the D flip-flop implementation by a quantity that can only be determined on a case-by-case basis.
- VII:** The number of commonalities induced by the rule is less than double those in the D flip-flop implementation by a quantity that can only be determined on a case-by-case basis.

(Part C) The final implementation to consider is that constructed with **JK flip-flops**. Again, because JK flip-flops double the number of inputs, one would expect that by using the 2 heuristics above, the commonalities in the bits of the next-state logic should double as well. However, further analysis might yield one of the cases enumerated below. **For each of the two aforementioned rules, please choose an option below and provide adequate justification** as to how the number of pairs with guaranteed commonalities in a JK flip-flop implementation compares with that of a D flip-flop implementation (the choices are identical to that in Part B). Your answers for the two rules may not be the same.

- I:** The number of commonalities induced by the rule is exactly double what is present in a D flip-flop implementation.
- II:** The number of commonalities induced by the rule exceeds double those in the D flip-flop implementation by 1.
- III:** The number of commonalities induced by the rule is less than double those in the D flip-flop implementation by 1.
- IV:** The number of commonalities induced by the rule exceeds double those in the D flip-flop implementation by a fixed, known quantity greater than 1 (but you don't need to report the exact quantity).
- V:** The number of commonalities induced by the rule is less than double those in the D flip-flop implementation by a fixed, known quantity greater than 1 (but you don't need to report the exact quantity).
- VI:** The number of commonalities induced by the rule exceeds double those in the D flip-flop implementation by a quantity that can only be determined on a case-by-case basis.
- VII:** The number of commonalities induced by the rule is less than double those in the D flip-flop implementation by a quantity that can only be determined on a case-by-case basis.