CSE 140 Homework Four

November 8, 2016

*Only **Problem Set Part B** will be graded. Turn in only **Problem Set Part B** which will be due on November 16, 2016 (Wednesday) at 3:00pm.*

*Submit homework via gradescope <gradescope.com> as a **PDF.** Other formats may or may not be viewed and evaluated accurately.*

1 Problem Set Part A

All questions in this part are from Roth&Kinney, 7th Edition.

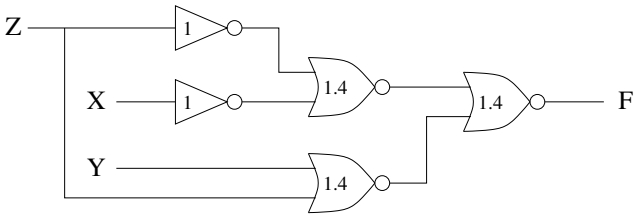
- 7.1, 7.4, 7.5, 7.6, 7.8, 7.9, 7.10, 7.11, 7.14, 7.26, 7.27, 7.28, 7.41, 7.42
- 8.2, 8.6, 8.7, 8.8
- 13.3(a)(b), 13.4(a), 13.8(a), 13.11(a)
- 14.4, 14.12, 14.13, 14.17, 14.23

2 Problem Set Part B

1 (Static Hazards)

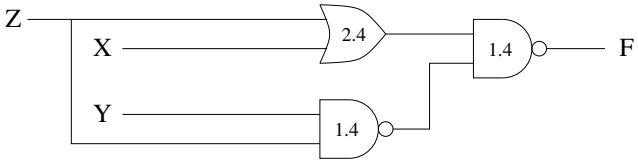
As we have studied in class, static hazards are caused by two complementary signals which exhibit identical values for short periods of time due to different delays on the various paths through the schematic. A **static 1-**hazard is observed if the output which should be stable at 1 has a glitch of 0, while a **static 0-**hazard is observed if the output which should be stable at 0 has a glitch of 1.

(Part A) For each of the circuits given below, please determine whether it will have a static hazard or not. If you think a function will have a static hazard, please determine the type of hazard (*static-0* or *static-1*), and the input transition(s) (e.g., XYZ: 000 → 001) that cause the hazard.



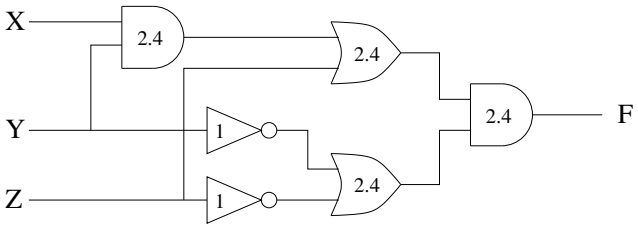
Hazard?	Type?

Input transition(s):



Hazard?	Type?

Input transition(s):



Hazard?	Type?

Input transition(s):

(Part B) For circuits that are implemented using only AND, OR, and NOR gates, static hazards can be detected using K-Maps. A **static 1**-hazard occurs if there exist two adjacent 1-minterms that are not covered by a common product term in a *sum-of-products* implementation.

In this part, you are asked to evaluate the possibility of static 1-hazards for the **minimum sum-of-products** implementations of 3-variable Boolean functions. As you know, a 3-variable Boolean function F can contain i ($i = 0...8$) 1-minterms. Obviously, functions of 0, 1, 7 and 8 minterms have no static hazards. The question you need to answer is as follows: given a function F that contains i ($i = 2...6$) 1-minterms, is it possible for its **minimum sum-of-products** implementation to exhibit a **static 1**-hazard?

If you think a possible Boolean function F with i minterms may have a static 1-hazard, please give an example of F in the K-map, show the minimal sum-of-products expressions, and provide the input transition(s) that cause the hazard. Otherwise, if you think it is impossible for the function under consideration to have a static 1-hazard, please explain your reason using K-maps.

A 2-minterm Boolean function: Static 1-hazard possible? **Yes / No**

$x\backslash yz$	00	01	11	10
0				
1				

A 3-minterm Boolean function: Static 1-hazard possible? **Yes / No**

$x\backslash yz$	00	01	11	10
0				
1				

A 4-minterm Boolean function: Static 1-hazard possible? **Yes / No**

$x\backslash yz$	00	01	11	10
0				
1				

A 5-minterm Boolean function: Static 1-hazard possible? **Yes / No**

$x\backslash yz$	00	01	11	10
0				
1				

A 6-minterm Boolean function: Static 1-hazard possible? **Yes / No**

$x\backslash yz$	00	01	11	10
0				
1				

(Part C) If a 3-variable function F has static-1 hazard(s) in its **minimum sum-of-products** implementation, will it also have static-0 hazard(s) in its **minimum product-of-sums** implementation? Please clearly mark your choice and justify your answer.

(A) It is impossible for F to have static-0 hazard(s) in its minimum product-of-sums implementation.

(B) F will always exhibit static-0 hazard(s) in its minimum product-of-sums implementation.

(C) It is possible for F to have static-0 hazard(s) in its minimum product-of-sums implementation, but not necessary.

Your selection is _____.

Your reasoning:

2 (Multi-Level Gate Circuits)

When using Karnaugh maps in this class, we generally are trying to find a two-level form of the function, either a sum-of-products or a product-of-sums. In this question we will explore Karnaugh maps and boolean functions in their three-level representation. For simplicity's sake, we will limit ourselves to OR-AND-OR forms (e.g. $(a+b)c' + (a'+b')(c+d)$). Notice that in this form we are applying an OR operation to two product-of-sums expressions.

(Part A) First, let's just look at a K-map and understand what a three-level form would look like. Since each term in a three-level form is actually a two-level form function of its own, we can decompose the function into two two-level form functions. In this question, let t_1 and t_2 be the decompositions of F (all of their K-maps are given below). Define each of the terms as a **product of sums** and write down what the function F is as a three-level boolean equation.

F

$ab\backslash cd$	00	01	11	10
00	1	1	0	0
01	1	0	1	0
11	1	0	1	1
10	1	1	0	0

t_1

$ab\backslash cd$	00	01	11	10
00	1	1	0	0
01	1	0	x	0
11	1	0	x	x
10	1	1	0	0

$+$

t_2

$ab\backslash cd$	00	01	11	10
00	x	x	0	0
01	x	0	1	0
11	x	0	1	1
10	x	x	0	0

$F = t_1 + t_2 =$

(Part B) Now let's go the other way. Take the function F below and fill out its K-map. Instead of circling implicants the way we have throughout this course, draw whatever shape is necessary to surround the two-level terms given. Make sure to label them as t_1 and t_2.

$$F = \underbrace{ab(c'+d)}_{t_1} + \underbrace{c(a'+b')}_{t_2}$$

$ab\backslash cd$	00	01	11	10
00				
01				
11				
10				

(Part C) Now, to understand how we can work with the product of sums, we will work with applying an AND operation on two K-maps. This time, you have two partially filled K-maps which, under the AND operation, output the function F. Fill the two out appropriately so that they output the proper result and write out the function F in terms of the two functions.

F

$ab\backslash cd$	00	01	11	10
00	1	1	0	0
01	1	1	0	0
11	1	0	0	0
10	1	1	0	0

t_1

$ab\backslash cd$	00	01	11	10
00	1	1		
01	1	1		
11	1	1		
10	1	1		

$\cdot$

t_2

$ab\backslash cd$	00	01	11	10
00			1	1
01			1	1
11			1	1
10			1	1

$F = t_1 \cdot t_2 =$

(Part D) Now let's bring these multi-level functions into the context of multi-output functions. By using multi-level functions rather than two-level functions, we can discover new terms that we wouldn't have been able to share between our functions otherwise. However, keep in mind that we could be sharing the implicates of a product-of-sums term, not the implicants, so the shared expressions between two K-maps can be less than obvious.

Take these two K-maps and try to find three-level representations for both of them so that a gate implementation would only require three OR gates and four AND gates. Write down these three-level boolean equations below.

Hint: Remember that you can decompose like we did in (Part A). Also, look for shared minterms within each of the components. If we were implementing the solution with a gate circuit, it should only use 7 gates.

F_1

$ab\backslash cd$	00	01	11	10
00	1	1	0	0
01	1	0	1	0
11	0	1	1	0
10	0	0	0	0

F_2

$ab\backslash cd$	00	01	11	10
00	0	1	0	0
01	1	0	1	1
11	0	1	0	0
10	0	1	0	0

$F_1 =$

$F_2 =$

(Part E) Finally, we'll play with more complicated functions and make a multi-level, multi-output gate circuit. Use what you've seen to find a three-level representation for each K-map below and then connect the given logic gates to implement the functions. Keep your wiring as neat as possible.

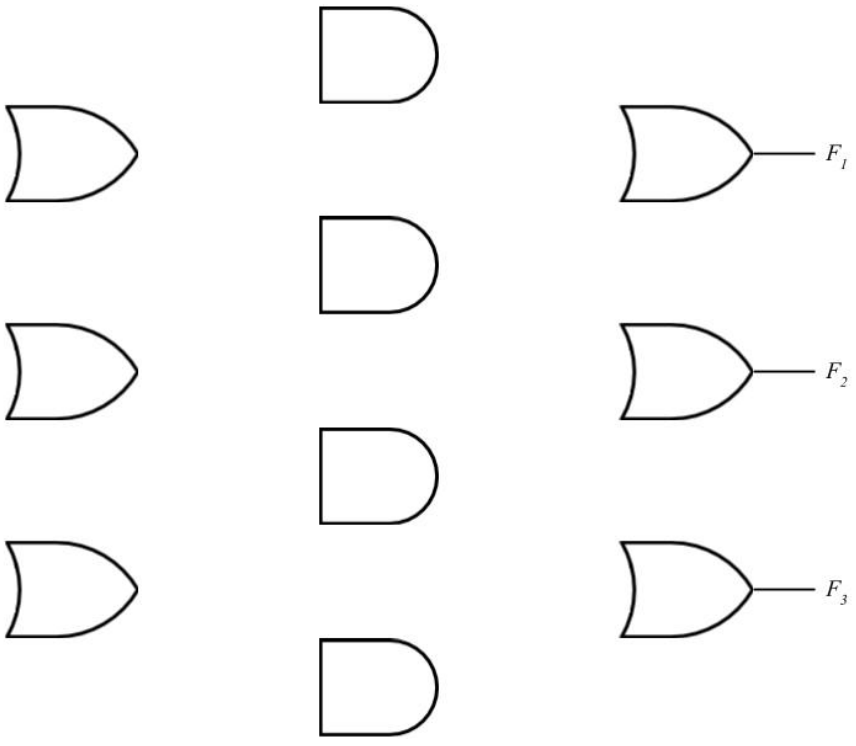
A few notes: Each level of gates can only connect to the next level of gates (these are OR-AND-OR implementations). Literals can be written next to any of the gates as inputs; you don't have draw wires coming from one source.

F_1

$ab\backslash cd$	00	01	11	10
00	1	0	0	0
01	1	0	1	1
11	1	1	1	0
10	1	1	0	0

F_2

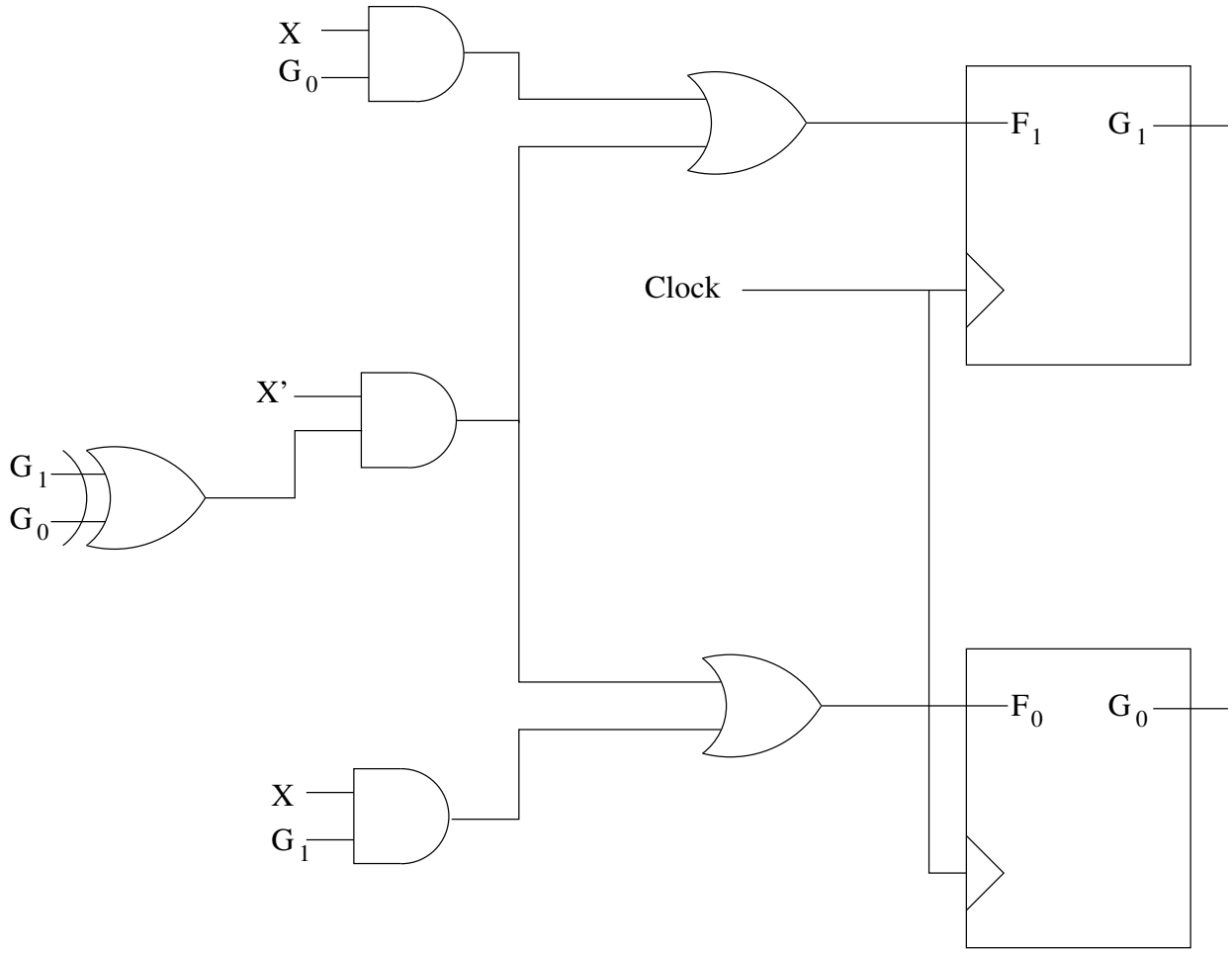
$ab\backslash cd$	00	01	11	10
00	0	0	1	1
01	0	0	1	1
11	0	1	1	0
10	0	1	1	1



3 (FSM Analysis)

Below is the circuit diagram for a finite state machine that your boss has given you to implement just before leaving for his vacation. He has told you that all four states that it may represent are in use–that is, between any two states p and q, there is a path either from p to q, from q to p, or both. However, as you can see, he neglected to record what types of flip-flops are used to encode the states!

(Part A) While staring confoundedly at the diagram and growing increasingly angry at your boss, you realize that it may be possible for you to narrow down what flip-flops your boss must have had in mind. Given the constraint that all states must be in some way connected as described above, determine which type of flip-flops G_1 and G_0 must be. If you think all combinations, no combinations, or some subset of combinations of flip-flop types are valid, say so. Note that, while consistency is often a good idea, there is no requirement that all flip-flops in a machine must be of the same type. *Hint: The fact that each has only one input limits your options!*



(Part B) In the space below, draw the state diagram for the FSM which uses the flip-flops you picked in (Part A). Clearly label state encodings. (If you think there are multiple possibilities that meet the constraints set forth in the beginning of this question, just pick one. If you think there are no such possibilities, draw the diagram for an errant machine and show why it does not meet the constraints.)