

CSE 140 Homework Three

March 3, 2014

*Only **Problem Set Part B** will be graded. Turn in only **Problem Set Part B** which will be due on March 14, 2014 (Friday) at 4:00pm.*

1 Problem Set Part A

- Roth&Kinney, 6th Ed: 7.1
- Roth&Kinney, 6th Ed: 7.4 - 7.6
- Roth&Kinney, 6th Ed: 7.8 - 7.9
- Roth&Kinney, 6th Ed: 7.15 (a)
- Roth&Kinney, 6th Ed: 8.2
- Roth&Kinney, 6th Ed: 8.6 - 8.8
- 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: 11.2
- Roth&Kinney, 6th Ed: 11.15 - 11.16
- Roth&Kinney, 6th Ed: 13.3(a)(b)
- Roth&Kinney, 6th Ed: 13.4(a)
- Roth&Kinney, 6th Ed: 13.8(a)
- 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

2 Problem Set Part B

5 (Static Hazards) (24 points)

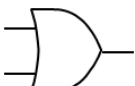
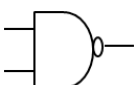
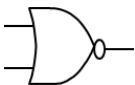
As you have learned in class, static hazards manifest themselves when adjacent minterms (or maxterms) are not covered by a common implicant (or implicate). For example, the function $(x + y)(x' + z')$ contains a static-0 hazard because maxterms $x + y + z'$ and $x' + y + z'$ fail to be covered by a common implicant, as shown in the K-map below, where one of the prime implicants in a minimal cover is bolded and the other italicized:

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

For the following parts, please refer to the Karnaugh map below when writing your answers.

wx\yz	00	01	11	10
00	0	1	1	0
01	0	1	0	0
11	1	1	0	0
10	0	1	1	0

You should also refer to the gate-delays printed below.

Name	Graphic Symbol	Functional Expression	Delay
Inverter		$F = x'$	1 ns
AND		$F = xy$	2.4 ns
OR		$F = x + y$	2.4 ns
NAND		$F = (xy)'$	1.4 ns
NOR		$F = (x + y)'$	1.4 ns

(Part A) As you can probably tell, the 4-variable function above contains at least one static hazard in its minimal product-of-sums implementation. Please draw a 2-level gate-level implementation of its minimal *product-of-sums* cover, and let us know the transitions (and the corresponding directions) in which each hazard manifests itself.

(Part B) One of your fellow students has suggested that a possible way to eliminate the static hazards in this circuit is to convert it from the minimal 2-level product-of-sums implementation to the corresponding minimal 2-level **NOR-NOR** implementation (of course in either case in addition to the two levels, inverters can appear at the inputs but not at the output side of the implementation). Do you agree that this transformation will fix the static hazard(s)? If so, please provide your reasoning below. If not, please indicate the hazards that will occur, as well as the directions in which they occur.

(Part C) Another one of your fellow students has suggested that another possible way to eliminate the static hazards in your circuit is to convert it from a minimal 2-level product-of-sums implementation to a minimal 2-level **NAND-NAND** implementation (of course in either case in addition to the two levels, inverters can appear at the inputs but not at the output side of the implementation). Do you agree that this will fix the static hazard(s)? If so, please provide your reasoning below. If not, please indicate the hazards that will occur, as well as the directions in which they occur.

(Part D) Another one of your fellow students has suggested that yet another possible way to eliminate static hazards in your circuit is to convert exactly *one* of the **OR-gates** in your 2-level minimal product-of-sum implementation to a **NAND-gate** (with the correct inverter propagations, of course), while leaving the other gates as they are in the product-of-sums implementation. Do you agree that this method will fix the static hazards? If you believe that swapping any one of your OR-gates with a NAND-gate will eliminate static hazards, please provide your reasoning below. If you believe that there are only certain OR-gates that can be swapped, please specify which one(s) can be swapped, and provide your reasoning below. If you believe that none of the OR-gates can be swapped to eliminate the static hazard, please provide your reasoning below.

2 (StaR Flip-Flops) After their wonderful performance at the PROM, Inhibition and Implication graduate from CircuitBoard High with newfound aspirations for becoming professional dancers. They want to be stars! Quickly after graduating, Inhibition joins a dance troupe called “The Flip-Flops”.

Not long after Inhibition joins the dance troupe, which is normally populated solely by NAND and NOR gates, the choreographer, Philippa, notices that Inhibition has some unique attributes that may help the troupe perform better than they normally would.

Looking at the troupe’s normal routine during the “SR-Flip-Flop” dance, and taking into consideration Inhibition’s unique talents, she realizes she can modify the dance so that it has the behavior described by the following characteristic table:

S^*	R^*	Q_{t+1}
0	0	Q_t
0	1	0
1	0	1
1	1	Q_t

Due to the improvement in performance that she believes this new dance exhibits, and the fame she anticipates it will bring to the dance troupe, she decides to rename the dance and call it the “StaR-Flip-Flop”. Note that the inputs are denoted S^* and R^* , so as to distinguish this Flip-Flop type from standard SR-Flip-Flops.

In this question, you will be exploring the properties of this new type of Flip-Flop.

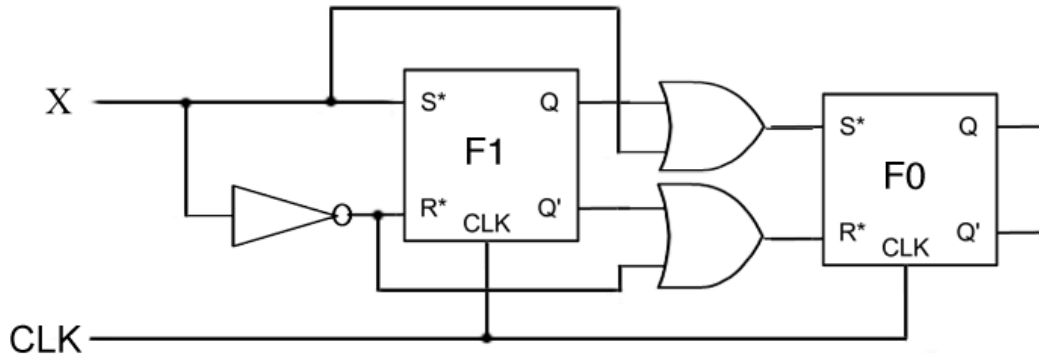
(Part A) Identify the characteristic equation for the StaR-Flip-Flop, in terms of S^* , R^* , and Q_t , where Q_t represents the current value of the Flip-Flop’s state (and Q_{t+1} represents the next value). (Examples of what characteristic equations should look like can be found in the back of your exam.) Furthermore, fill out the excitation table for the StaR-Flip-Flop below by listing **all** the possible input values for S^* and R^* which result in the transition shown in each row.

$Q_{t+1} =$ _____

StaR-Flip-Flop Excitation Table:

$Q_t \rightarrow Q_{t+1}$	$S^* R^*$ Input Combinations
0 0	
0 1	
1 0	
1 1	

(Part B) To give the dancers some practice with the techniques that Inhibition introduces, Philippa arranges the following practice routine, called “The StaR-Boot-Slide”, for the dancers to carry out:



Derive the state-transition behavior for this FSM. Start by using the S^* and R^* logic for both of the Flip-Flops, and then using those to figure out the next-state logic for each Flip-Flop. Finally, fill out the state transition table. (Note that because this is just a practice dance, Philippa has not specified any output behavior for this dance, since nobody will actually be watching the troupe perform it.)

S_0^* _____

R_0^* _____

S_1^* _____

R_1^* _____

$Q_{0(next)}$ _____

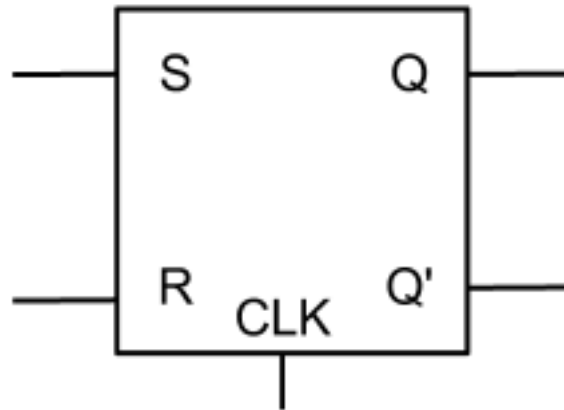
$Q_{1(next)}$ _____

Current State		X = 0	X = 1
Q_1	Q_0		
0	0		
0	1		
1	0		
1	1		

(Part C) On the second day of rehearsal, Inhibition suddenly comes down with a virus, and has to call in sick. In order for regular rehearsal to continue, you must help Philippa to explain how to implement the dance using ordinary SR-Flip-Flops with optimal next-state logic. In the space below, using SR-Flip-Flops, draw the circuit which mimics the StaR-Flip-Flop implementation of the FSM in Part B. The state encodings and transition behavior should be identical to the StaR-Flip-Flop implementation.

(Part D) Some of Philippa's contemporaries have picked up on rumors that one of her new students shows promise and may very well have already revolutionized the way dances are performed. They are trying to guess how Philippa is using Inhibition to implement the StaR-Flip-Flop.

Below is an ordinary SR-Flip-Flop. Add Inhibition gates to the diagram, and label the inputs S^* and R^* , so that the resulting circuit behaves as described in the introduction to this problem.



3 (State Un-Minimization) Under ordinary circumstances, it is beneficial to minimize the state-space of an FSM, according to the algorithm you were taught in class. In this question, you will be investigating instances where a non-minimal state-space for an FSM is actually better than the minimal implementation.

(Part A) Suppose we want to make a **No-Rollover Base-5 Incrementer**, which takes a base-5 digit x and outputs $x + 1$, **unless $x = 4$, in which case instead of rolling over to 0, it simply outputs 4**. Suppose further that we have encoded each of the digits 0-4 with their 3-bit binary counterparts. Find the boolean functions in minimal sum-of-products form for each of the output bits O_0 , O_1 , and O_2 .

O_0	x_1x_0	00	01	11	10
x_2					
0					
1					

O_1	x_1x_0	00	01	11	10
x_2					
0					
1					

O_2	x_1x_0	00	01	11	10
x_2					
0					
1					

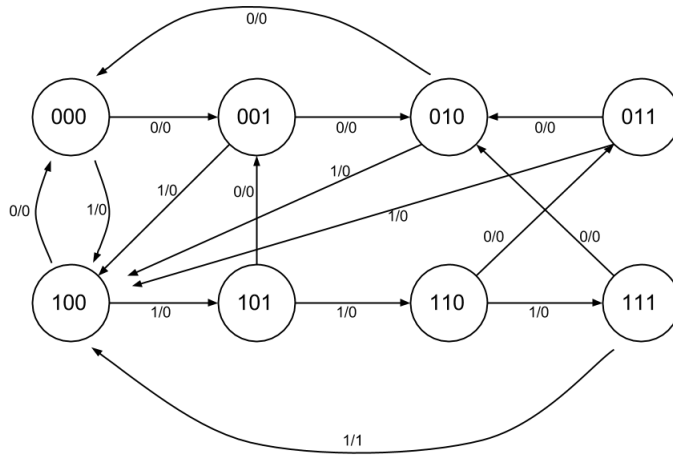
(Part B) As you should have seen in Part A, the fact that there were many unused input/output values induced don't-cares in many spaces in the Karnaugh Maps for the functions O_0 , O_1 , and O_2 . However, there is another way to induce don't-cares that results in a different distribution of them in the Maps. This is to loosen encoding for the digit "4". That is, **instead of representing 4 with the single encoding "100", 4 can be represented by any of the encodings unused by the digits 0-3**. Now that the number 4 can be represented by multiple bit combinations, you realize that you are going to lose some of the original don't care's in your Karnaugh maps, but perhaps you can compensate for that loss by having the inputs that produce the value 4 map to the best bit combination to help you obtain the smallest sum of products implementation. Fill in the Karnaugh Maps for the functions O_0 , O_1 , and O_2 using this encoding scheme and derive the minimal sum of products representation.

O_0	x_1x_0	00	01	11	10
x_2					
0					
1					

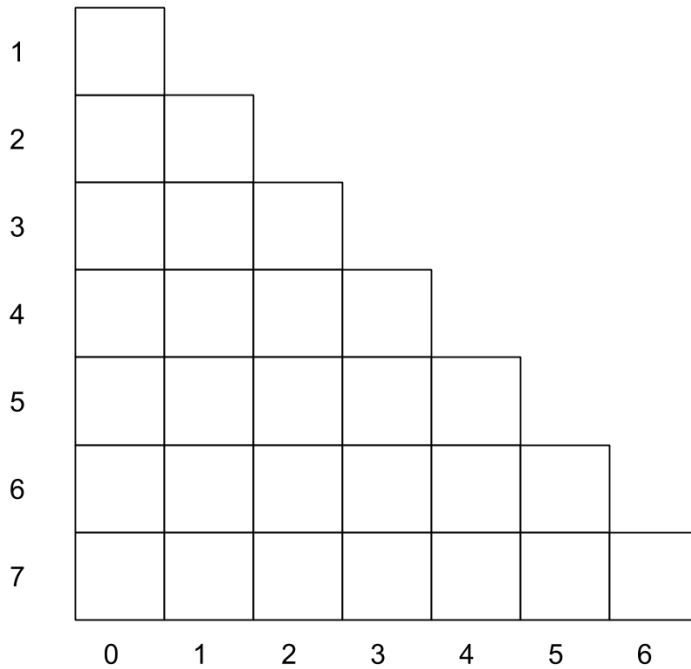
O_1	x_1x_0	00	01	11	10
x_2					
0					
1					

O_2	x_1x_0	00	01	11	10
x_2					
0					
1					

(Part C) The Mealy FSM below has one input, I , and one output, O . Fill in the implication table, and identify the final equivalence classes among the states. Then draw the minimized state transition diagram.



State	$I = 0$	$I = 1$
000	001 / 0	100 / 0
001	010 / 0	100 / 0
010	000 / 0	100 / 0
011	010 / 0	100 / 0
100	001 / 0	101 / 0
101	001 / 0	110 / 0
110	011 / 0	111 / 0
111	010 / 0	100 / 1



(Part D) Find the next-state logic for each of the Flip-Flops of the minimized FSM you produced in part C. Assume the FSM is implemented with D-Flip-Flops.

(Part E) Normally, state minimization induces don't-cares in the next-state logic, since there are unused states whose encodings can be don't-cares. However, similarly to Part B, there is a way of loosely encoding one of the states in the minimized FSM you produced in Part C, which will result in a different distribution of don't-cares in the next-state logic. While state minimization does reduce flip flops and does increase don't-cares in the associated FSM, *redundant* states can perhaps minimize the logic expressions as explored in a combinational framework in Part B. You can use Part C as an inspiration to developing an equivalent FSM with redundant state(s) which will end up minimizing the next state logic. Please write the possibly redundant FSM and its minimal sum of products next state logic implementation with **D** Flip Flops.

4 (PLA Optimizations) When optimizing PLA implementations of boolean functions, effort is largely driven towards minimizing the number of rows that are used. While for single-output functions, the best that can be done in this regard is the application of the Quine-McCluskey tabulation method we learned in class, for multi-output functions, more creative explorations in the space of implicants, perhaps by even sacrificing primality, can deliver further minimization benefits. Additional flexibility is obtained through the use of XOR gates at the outputs that enable one to combine the minterms of one function with the maxterms of another, should sharing be more fruitful in such a manner.

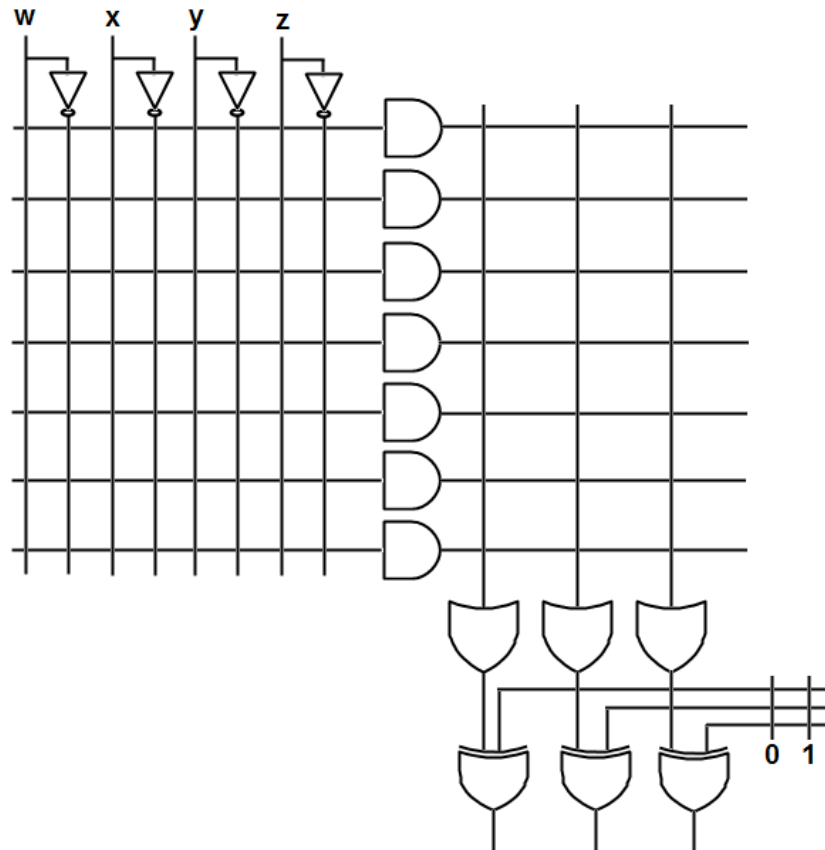
(Part A) Please use the aforementioned optimization technique to generate the optimal PLA for the following 2 functions, F_0 and F_1 . You should need no more than 5 rows.

$$F_0 =$$

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

$$F_1 =$$

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



(Part B) One can extend the PLA to account for additional functionality. For example, as you have probably noticed, an XOR decomposition of the aforementioned two functions can help tremendously in simplifying the number of minterms that your PLA requires. Before we consider the PLA optimizations we can make, please fill in the K-maps below with XOR decompositions of the two functions that enable a simplification of the implicant logic. For your convenience, the functions are shown again below.

$$F_0 = \begin{array}{c|cccc} wx \backslash yz & 00 & 01 & 11 & 10 \\ \hline 00 & 0 & 0 & 0 & 1 \\ \hline 01 & 0 & 0 & 0 & 1 \\ \hline 11 & 0 & 0 & 0 & 1 \\ \hline 10 & 1 & 1 & 1 & 0 \end{array} =$$

$wx \backslash yz$	00	01	11	10
00				
01				
11				
10				

$\oplus$

$wx \backslash yz$	00	01	11	10
00				
01				
11				
10				

$$F_1 = \begin{array}{c|cccc} wx \backslash yz & 00 & 01 & 11 & 10 \\ \hline 00 & 1 & 1 & 1 & 0 \\ \hline 01 & 0 & 0 & 1 & 0 \\ \hline 11 & 0 & 0 & 1 & 0 \\ \hline 10 & 0 & 0 & 0 & 1 \end{array} =$$

$wx \backslash yz$	00	01	11	10
00				
01				
11				
10				

$\oplus$

$wx \backslash yz$	00	01	11	10
00				
01				
11				
10				

(Part C) We can now extend the PLA to account for the decompositions that you have completed in (Part B). You might have noticed that (assuming your implementation is minimal), entire outputs of a single decomposition can be shared. As such, one can modify the PLA to contain an extra row of XOR gates at the output, where the inputs for each XOR can consist of either a single output (which would be denoted by putting a dot over the intersection between the second input and 0) or 2 adjacent outputs (which would be denoted by putting a dot over the intersection between the output of the XOR on the top row and the input of the XOR at the bottom row). As you have probably guessed, this enables your PLA implementation to actually consist of an XOR of your adjacent outputs, which helps with your XOR decomposition. Keeping these heuristics in mind, please fill in the dots on the PLA below so as to utilize the minimal number of rows while also taking advantage of your XOR decomposition from (Part B). Your solution should require no more than 4 rows.

