Midterm Examination

Your Name: *Solutions* Student Id #:

Please make sure to read the following instructions carefully and in entirety:

1. You may use your textbook plus one original handwritten cheat sheet of a two-sided page.
2. You may not use any electronic device.
3. You must show the intermediate steps in deriving your answer.
4. You must answer in the space provided in this booklet, and not use any other sheets.
4. If any sheets are missing from this exam booklet, you will get zero grade on the entire exam.

Following table to be filled by course staff only

Problem #	Maximum Score	Your Score	Comments
1	10		
2	10		
3	10		
4	15		
5	15		
TOTAL	60		

IMPORTANT

Please do not tear off or remove any pages from this exam booklet. You must return all the pages of this booklet, and failure to do so will void your exam.

This solution file available at http://goo.gl/U9Bzdo

Problem #1 [5 * 2 = 10 points]

Place an X to the left of the correct choice in the following multiple choice questions. There is space on the next page for you to work out the answers in case you need it.

a) Simplify the Boolean expression $((A \vee B \vee C) \wedge \neg(D \vee E)) \vee ((A \vee B \vee C) \wedge (D \vee E))$ and choose the best answer.

1. $A \vee B \vee C$ **CORRECT**
2. $D \vee E$
3. $\neg A \wedge \neg B \wedge \neg C$
4. $\neg D \wedge \neg E$
5. None of the above

b) Which of the following relationships represents the dual of the Boolean property $x \vee (\neg x \wedge y) = x \vee y$?

1. $\neg x \wedge (x \vee \neg y) = \neg x \wedge \neg y$
2. $x \wedge (\neg x \wedge y) = x \wedge y$
3. $x \wedge \neg x \vee y$
4. $\neg x \wedge (x \wedge \neg y) = \neg x \wedge \neg y$
5. $x \wedge (\neg x \vee y) = x \wedge y$ **CORRECT**

c) Given the function $f(X, Y, Z) = (X \wedge Z) \vee (Z \wedge (\neg X \vee (X \wedge Y)))$, the equivalent most simplified Boolean representation for $f(X, Y, Z)$ is:

1. $Z \vee (Y \wedge Z)$
2. $Z \vee (X \wedge Y \wedge Z)$
3. $X \wedge Z$
4. $X \vee (Y \wedge Z)$
5. None of the above **CORRECT:** $f(X, Y, Z) = Z$

d) Simplification of the Boolean expression $(\neg(A \vee B) \wedge \neg(C \vee D \vee E)) \vee \neg(A \vee B)$ yields which of the following results?

1. $A \vee B$
2. $\neg A \wedge \neg B$ **CORRECT**
3. $C \vee D \vee E$
4. $\neg C \wedge \neg D \wedge \neg E$
5. $\neg A \wedge \neg B \wedge \neg C \wedge \neg D \wedge \neg E$

e) Given that $F = (\neg A \wedge \neg B) \vee \neg C \vee \neg D \vee \neg E$, which one of the following represents the correct expression for $\neg F$?

1. $\neg F = A \vee B \vee C \vee D \vee E$
2. $\neg F = A \wedge B \wedge C \wedge D \wedge E$
3. $\neg F = A \wedge B \wedge (C \vee D \vee E)$
4. $\neg F = (A \wedge B) \vee \neg C \vee \neg D \vee \neg E$
5. $\neg F = (A \vee B) \wedge C \wedge D \wedge E$ **CORRECT**

Problem #2 [10 points]

Determine the minimum-cost PoS expression and correspond circuit using NOR gates for the function $\sum m(4,6,8,10,11,12,15) + D(3,5,7,9)$. Assume the input variables are labeled x3, x2, x1, and x0, and the minterms are numbered with x3 in the most significant position and x0 in the least significant position (e.g. minterm 1 corresponds to the product term $\neg x3 \wedge \neg x2 \wedge \neg x1 \wedge x0$). Show all the necessary intermediate steps. Note that $D(\,)$ represents the don't care set.

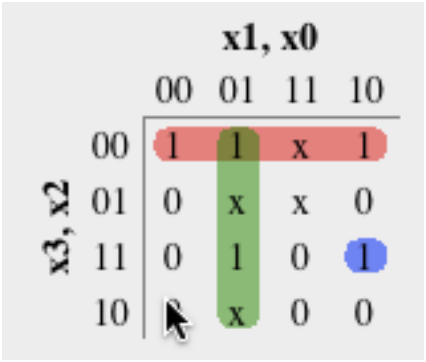
Answer #2:

The problem requires NOR gates, which indicates that we seek a PoS form.

APPROACH 1: We will develop a SoP for the complement of the function and then use De Morgan's theorem to get the PoS for the original function.

The complement of the function is $\sum m(0,1,2,13,14) + D(3,5,7,9)$.

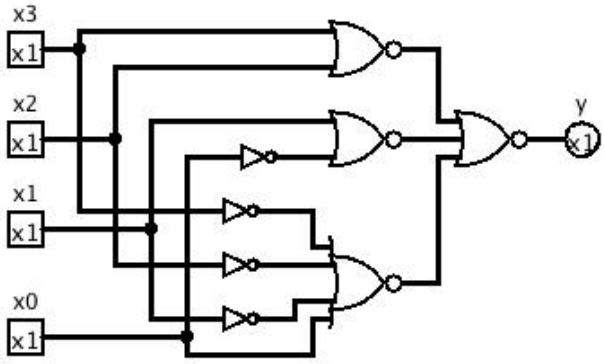
This gives the following K-Map, with the minimal cover shown:



The K-Map cover yields the following expression for the complement of the function:
$(\neg x3 \wedge \neg x2) \vee (\neg x1 \wedge x0) \vee (x3 \wedge x2 \wedge x1 \wedge \neg x0)$

Using duality theorem which states that the complement of the dual of a function evaluated over the complements of the inputs is the same as the original function over the original inputs, we get the following expression for the original function
$(x3 \vee x2) \wedge (x1 \vee \neg x0) \wedge (\neg x3 \vee \neg x2 \vee \neg x1 \vee x0)$ which using De Morgan's theorem can be rewritten as $\neg(\neg(x3 \vee x2) \vee \neg(x1 \vee \neg x0) \vee \neg(\neg x3 \vee \neg x2 \vee \neg x1 \vee x0))$

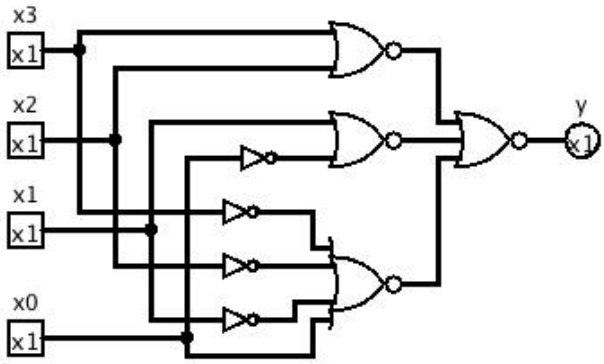
Answer #2 (contd.):



APPROACH 2: We can directly work with the K-Map of the original function but focus on entries with 0 and directly go to product of sum form.

x3, x2 \ x1, x0	00	01	11	10
00	0	0	x	0
01	1	x	x	1
11	1	0	1	0
10	1	x	1	1

$(x3 \vee x2) \wedge (x1 \vee \neg x0) \wedge (\neg x3 \vee \neg x2 \vee \neg x1 \vee x0)$ which using De Morgan's theorem can be rewritten as $\neg(\neg(x3 \vee x2) \vee \neg(x1 \vee \neg x0) \vee \neg(\neg x3 \vee \neg x2 \vee \neg x1 \vee x0))$ and yields the circuit



Problem #3 [10 points]

Implement the following function with inputs $w1$, $w2$, $w3$, and $w4$:

$f(w1, w2, w3, w4) = (w1 \wedge w2 \wedge w4) \vee (w1 \wedge w2) \vee (w1 \wedge w3) \vee (w1 \wedge w4) \vee (w3 \wedge w4)$

You have two options for this question.

Easy half credit option (i.e. maximum score you'd receive will be 5 out of 10): implement using an 8-to-1 multiplexer and as few other gates as possible from the following types of gates: inverter, 2-input NAND, and 2-input NOR.

Harder full credit option (i.e. maximum score you'd receive will be 10): implement using a 4-to-1 multiplexer and as few other gates as possible from the following types of gates: inverter, 2-input NAND, and 2-input NOR.

Answer #3:

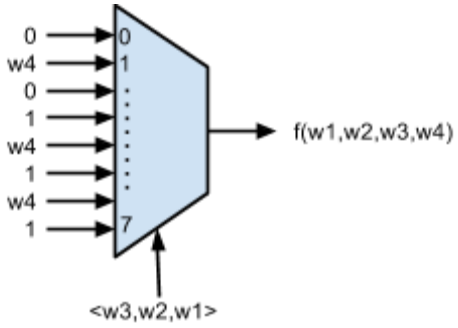
Easy half credit option:

Our strategy is to use 3 of the 4 inputs to control the mux, and use the remaining one to influence the data inputs to the mux. Several different solutions are possible depending on which inputs you select. Let us say we use $< w3, w2, w1 >$ as the control input $sb[2:0]$ of the 8-to-1 mux. Then the following table helps us derive how the data input lines of the mux should be connected.

$w3$	$w2$	$w1$	$f(w1, w2, w3, 0)$	$f(w1, w2, w3, 1)$	$f(w1, w2, w3, w4)$
0	0	0	0	0	0
0	0	1	0	1	$w4$
0	1	0	0	0	0
0	1	1	1	1	1
1	0	0	0	1	$w4$
1	0	1	1	1	1
1	1	0	0	1	$w4$
1	1	1	1	1	1

Answer #3 (contd.):

This yields the following circuit which only needs the 8-to-1 mux:



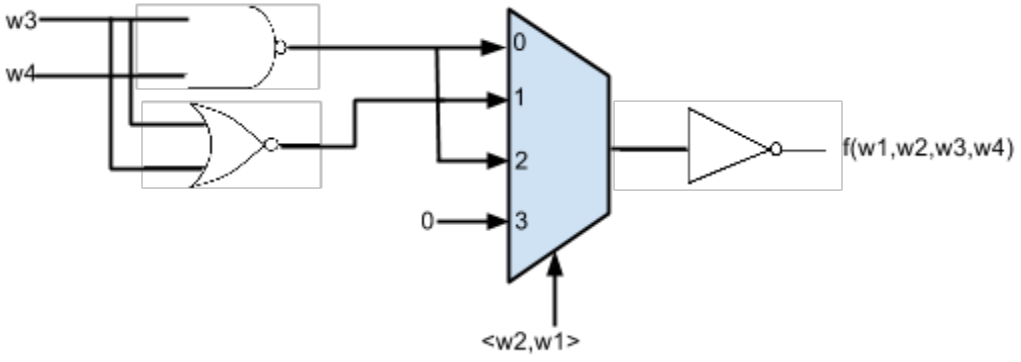
Harder full credit option

Our strategy is to use 2 of the 4 inputs to control the mux, and use the remaining one to influence the data inputs to the mux. Several different solutions are possible depending on which inputs you select. Let us say we use $< w2, w1 >$ as the control input $sb[1:0]$ of the 4-to-1 mux. Then the following table helps us derive how the data input lines of the mux should be connected.

$f(w1, w2, w3, w4) = (w1 \wedge w2 \wedge w4) \vee (w1 \wedge w2) \vee (w1 \wedge w3) \vee (w1 \wedge w4) \vee (w3 \wedge w4)$

$w2$	$w1$	$f(w1, w2, 0, 0)$	$f(w1, w2, 1, 0)$	$f(w1, w2, 0, 1)$	$f(w1, w2, 1, 1)$	$f(w1, w2, w3, w4)$
0	0	0	0	0	1	$w3 \wedge w4$
0	1	0	1	1	1	$w3 \vee w4$
1	0	0	0	0	1	$w3 \wedge w4$
1	1	1	1	1	1	1

This yields the following circuit which needs the 4-to-1 mux, one 2-input NAND, one 2-input NOR, and one inverters. Note that your circuit may look different if you selected different signals as mux control inputs.



Problem #4 [15 points]

Computational algorithms often involve shifting or rotating the bits of a variable by one or more bits to the left or to the right. Design a circuit that takes a 4-bit number *din[3:0]* and a 2-bit control signal *op[1:0]*, and outputs a 4-bit number *dout[3:0]* as follows:

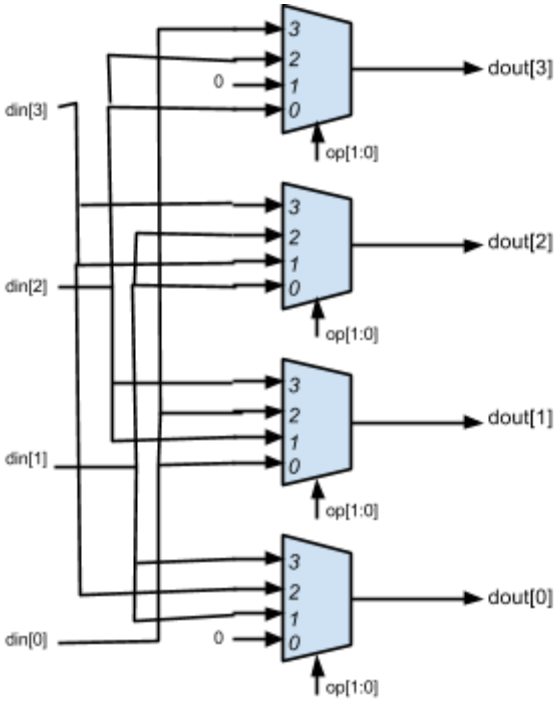
- if *op[1:0]==00* then shift left by 1 so that *dout* has the contents of *din* shifted left by *1* position with a 0 entering into the least significant bit position, i.e. d*out[3:1] = din[2:0]*, and *dout[0] = 0*.
- if *op[1:0]==01* then shift right by 1 whereby *dout* has the contents of *din* shifted right by 1 position with a 0 entering into the most significant bit position, i.e. *dout[2:0] = din[3:1]*, and *dout[3] = 0*
- if *op[1:0]==10* then rotate left by 1 whereby *dout* has the contents of *din* rotated left by 1 position, i.e. *dout[3:1] = din[2:0]*, and *dout[0] = din[3]*
- if *op[1:0]==11* then rotate right by 1 whereby *dout* has the contents of *din* rotated right by 1 position, i.e. *dout[2:0] = din[3:1]*, and *dout[3] = din[0]*

You may use any gates and building block modules discussed in the lectures, as well as wiring modules such as bus splitters. Clearly label the terminals of any building block modules you user.

Hint: Notice that shifting and rotating are really wiring operations, and don't need any logic gates.

Answer #4:

This is quite a simple system, just involving taking care of wiring details.



Problem #5 [15 points]

Digital systems often need to send data back to back over the same set of wires. The energy spent in sending a new data word depends on how many bits are different from the last data item that was sent over those wires. For example, say on 4-bit bus the system sent the word 0101 at time instant *t*, and then sends 0011 at time instant *t+1*, then two bits had changed (at positions 1 and 2). You have to design a 8-bit circuit that tries to save power by sending data or its complement, whichever will result in fewer bits changing relative to the last data item that was sent. To indicate whether the data is being sent complemented or not, an additional polarity bit is also sent on another wire along with the data, as described below.

Your circuit has three inputs (i) DIN[7:0] which is new data word to sent, (ii) PREV_DOUT[7:0] which are the bits that were sent to the output DOUT[7:0] in the previous transmission, and (iii) PREV_POLARITY, which represents the polarity bit value sent in the previous transmission. The circuit has two outputs (i) DOUT[7:0], which is the data bits being sent and will be either DIN[7:0] or ¬DIN[7:0], and (ii) POLARITY, which indicates the polarity of the data bits and is 0 if the data bits being sent correspond to DIN[7:0], and 1 if they correspond to ¬DIN[7:0].

To determine what to output on DOUT[7:0] and POLARITY, the algorithm is as follows:

- Let *n* be the # of bit positions at which DIN[7:0] and PREV_DOUT[7:0] differ.
- Now if PREV_POLARITY is 1, then sending DIN[7:0] on DOUT[7:0] and 0 on POLARITY would mean *m=n+1* wires out of 9 wires (8 for data and 1 for polarity) will see bit flips.
- On the other hand if PREV_POLARITY is 0, then sending DIN[7:0] on DOUT[7:0] and 0 on POLARITY would mean *m=n* wires out of 9 wires (8 for data and 1 for polarity) will see bit flips.
- Now, if *m<5* then the system should output DOUT[7:0]=DIN[7:0] and POLARITY=0 as that will result in fewer wires seeing change of state, and otherwise it should output DOUT[7:0]=¬DIN[7:0] and POLARITY=1.

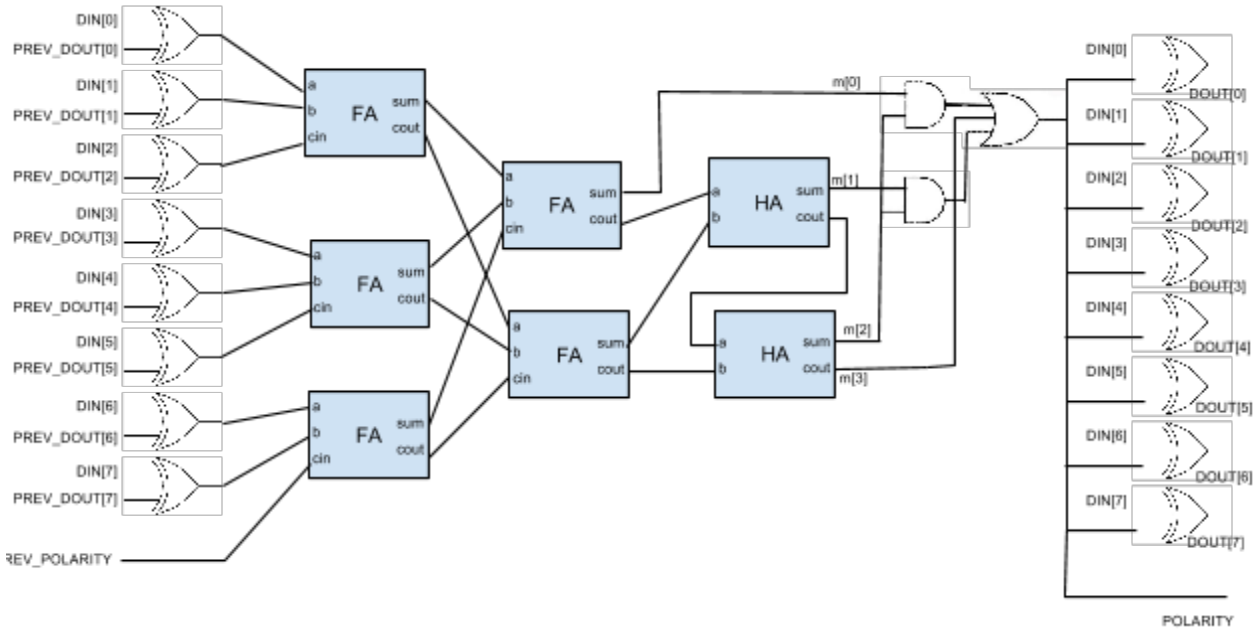
So for example, if PREV_DOUT[7:0] = 01101000, PREV_POLARITY=1, and DIN[7:0] = 01010100 then sending with positive polarity will mean DOUT[7:0] = 01010100 and POLARITY = 0, thus resulting in 5 out of 9 wires changing states. So it will be better to send the complement, i.e. DOUT[7:0] = ¬01010100 = 10101011 and POLARITY=1.

You may use any gates and building block modules discussed in the lectures. Clearly label the terminals of any building block modules that you use.

Hint: first derive an 8-bit intermediate variable that has 1 in those bit positions where DIN[7:0] and PREV[7:0] are different (what gate will use for detecting two bits are different?). Then count the number of 1s in this intermediate variable. Recall, such a counting circuit was done in Problem Set #2 using adders. Start by thinking about the algorithm behind your design, and then map it to building blocks and gates.

Answer #5:

Let us first think of the strategy. To find *n*, the # of bit positions at which DIN[7:0] and PREV_DOUT[7:0] differ, we need to count the # of 1s in DIN[7:0] XOR PREV_DOUT[7:0] since an XOR gate outputs 1 when the two inputs are different. Next, to find *m*, we need to compute *n* + PREV_POLARITY. In other words, we need to count the # of 1s in the 8 bits of (DIN[7:0] XOR PREV_DOUT[7:0]) and PREV_POLARITY. This can be done using a bunch of FAs, and will result in 4-bits of *m* whose value can be as much as 9 since we are counting # of 1s in a total of 9 bits. Next, if *m<5* then we need to output DIN[7:0] else its complement. While *m<5* could be accomplished using a 4-bit magnitude comparator module, it is much better to recognize that *m<5* occurs when either *m[3]==0 and m[2]==0* or when m[3:0]==0100, which after simplification using boolean logic can be implemented using a few gates as $m3 \vee (m2 \wedge m0) \vee (m2 \wedge m1)$. The necessary conditional complementation of DIN[7:0] can be done via eight XOR gates, one for each bit, since an XOR gate complements an input whenever the other input is logic-1. All of this yields the circuit below.



Try the preceding circuit in Logisim to convince yourself that this works! Note that there are other implementations possible too around the same basic idea. Note that some points will be deducted for solutions that may be functionally correct, but are not efficient. For example, points will be dedcuated for using a magnitude comparator (here we are comparing with a constant, and so magnitude comparator is a gross over-kill), or using inverter+mux for implementing the conditional complementation needed at the output stage (just use XOR), or not recognizing that comparing two bits being not equal can be done just via an XOR gate.