King Fahd University of Petroleum and Minerals

College of Computer Science and Engineering

Computer Engineering Department

COE 202: Digital Logic Design (3-0-3)
Term 181 (Fall 2018)
Major Exam 2
Saturday, November 11th, 2018

Time: 120 minutes, Total Pages: 9

Name:_____ ID:_____ Section: _____

Notes:
Do not open the exam book until instructed
Calculators are not allowed (*basic, advanced, cell phones, etc.*)
Answer all questions
All steps must be shown
Any assumptions made must be clearly stated
Clearly label all inputs and outputs of any circuit component

Question	Maximum Marks	Your Marks
1	**18**	
2	**19**	
3	**20**	
4	**13**	
Total	**70**	

Question 1. **(18 Marks)**

a) Consider the following Boolean function and its don't care conditions:

$$F(A,B,C,D) = \prod M(3,11,12,13) + \sum d(2,6,10,14)$$

Represent F on a K-map along with its don't care conditions. **(4 Marks)**

AB \ CD	00	01	11	10
00	1	1	0	X
01	1	1	1	X
11	0	0	1	X
10	1	1	0	X

b) The K-map below represents a function G(X,Y,Z,W):

XY \ ZW	00	01	11	10
00	1	1	0	1
01	1	X	X	1
11	0	X	X	0
10	1	1	0	0

1. List all the **prime implicants** and **essential prime implicants** of G. **(4 Marks)**

PIs: X'Z', X'Y, Z'W, Y'Z', X'W' EPIs: Y'Z' and X'W'

Note that Z'W is not EPI because only the don't care is not in any other PI – Don't cares do not have to be covered so they don't make an EPI!

2. Minimize G to a minimal SOP expression **(2 Marks)**

G = Y'Z' + X'W'

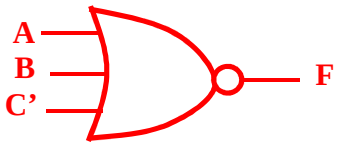
3. Minimize G to a minimal POS expression **(4 Marks)**

G = (Z' + W') (X' + Y') (X'+Z')

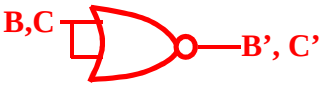
XY \ ZW	00	01	11	10
00	1	1	0	1
01	1	X	X	1
11	0	X	X	0
10	1	1	0	0

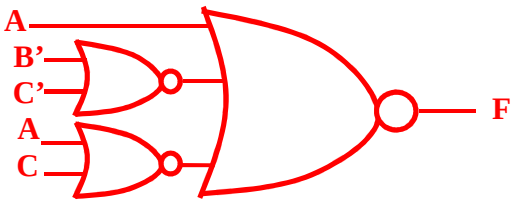
c) Let F = A' (B' + C') (A+C) Implement F using minimum number of NOR gates **only**. **(4 Marks)**

F is actually = m1 = A'B'C = (A + B + C')



Another solution (you lose 1 mark!)





Question 2: **(18 Marks)**

It is required to design a circuit that receives **a 3-bit signed** number in **signed-magnitude** representation, **X**, and computes the equation **Y = 3*X – 2** where **Y** is also **represented in signed-magnitude** representation.

a. Determine the number of bits needed for the output Y. Justify your answer. **(2 Marks)**

The smallest negative value is -3. Thus, 3*-3 -2 = -11. This implies that we need **5 bits** to represent the output correctly (1-bit for the sign, and 4-bits for the magnitude).

b. Fill the table below with the truth table for the output Y. **(5 Marks)**

X2 X1 X0	**Decimal Value of Y**	**Output Y:** Y4 Y3 Y2 Y1 Y0
0 0 0	-2	1 0 0 1 0
0 0 1	+1	0 0 0 0 1
0 1 0	+4	0 0 1 0 0
0 1 1	+7	0 0 1 1 1
1 0 0	-2	1 0 0 1 0
1 0 1	-5	1 0 1 0 1
1 1 0	-8	1 1 0 0 0
1 1 1	-11	1 1 0 1 1

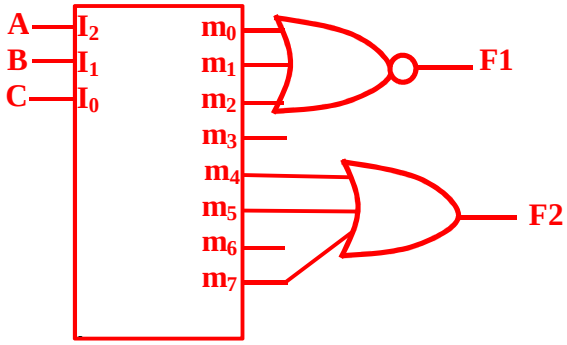
c. If the output of the circuit Y is fed as input to another circuit that will compute the equation: **$Z = Y^2$,** what will be the don't care conditions for **Z.** **(2 Marks)**

The don’t care conditions include all the values that do not occur at the output of Y, i.e. 0, +2, +3, +5, +6, +8 to +15, -1, -3, -4, -6, -7, -9, -10, -12, -13, -15, -16

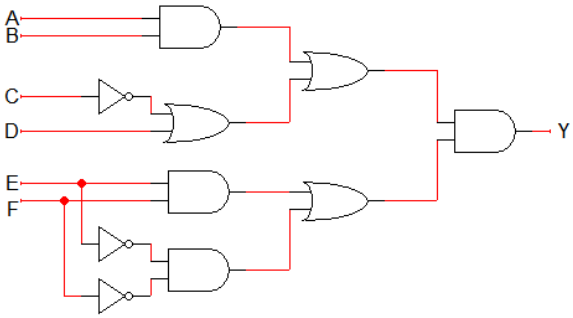
d. Implement the following two functions using **a decoder** and **additional gates, each** with the **smallest possible Fan-in (i.e. number of inputs). Clearly mark all inputs and outputs of the decoder.** F1 (A,B,C) = ∑m(0,4,5,6,7) F2 (A,B,C) = ∑m(4,5,7) **(5 Marks)**

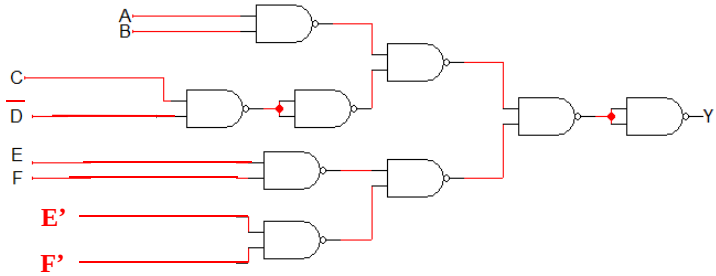
F1 (A,B,C) = ∑m(0,4,5,6,7) = ∏M(1,2,3) → it is better to implement it as POM
3 I/Ps → 3-to-8 decoder & a 3-I/Ps NOR Gate

F2 (A,B,C) = ∑m(4,5,7) = ∏M(0,1,2,3,6) → it is better to implement it as SOM
3 I/Ps → 3-to-8 decoder & a 3-I/Ps OR Gate



e. Convert the circuit below to **2-input NAND gates only.** Redraw the circuit to obtain a multi-level NAND circuit implementation. Assume that **both the true and complement** forms of each input variable are available. **Use the circuit as is and do not attempt to simplify it. (5 Marks)**





Question 3. **(20 Marks)**

a) Represent the given signed numbers using **8-bit** signed binary numbers in the specified representations in the table below: **(6 Marks)**

Decimal Value	Sign-Magnitude representation	1's complement representation	2's complement representation
-34	**1 010 0010**	**1101 1101**	**1101 1110**
-67	**1 100 0011**	**1011 1100**	**1011 1101**

b) **A** and **B** below are 8-bit signed integers represented in the 2's complement representation:

A = 11011000 B = 01110101

1. What are the decimal values of A and B? **(2 Marks)**

   Decimal Value of A = -128 + 64 + 16 + 8 = -40
   Decimal Value of B = 64 + 32 + 16 + 4 + 1 = +117

2. Compute A + B in binary and indicate whether there is overflow or not. **(3 Marks)**

```
1 1 1 1
  1 1 0 1 1 0 0 0
+ 0 1 1 1 0 1 0 1
------------------
  0 1 0 0 1 1 0 1
```

   There is NO overflow because C_7 and C_8 are identical. Last carry is ignored.

   The result = +77

3. Compute A – B by converting subtraction into binary addition. Indicate whether there is overflow or not. Show all the steps. **(3 Marks)**

```
A - B = A + (2's complement of B) = A + (1's complement of B) + 1
1     1 1       1
  1 1 0 1 1 0 0 0
+ 1 0 0 0 1 0 1 0
------------------
  0 1 1 0 0 0 1 1
```

Yes, there is overflow because C_7 and C_8 are different. The result is positive, while it is supposed to be negative. The exact result -157 cannot be represented using 8 bits only.

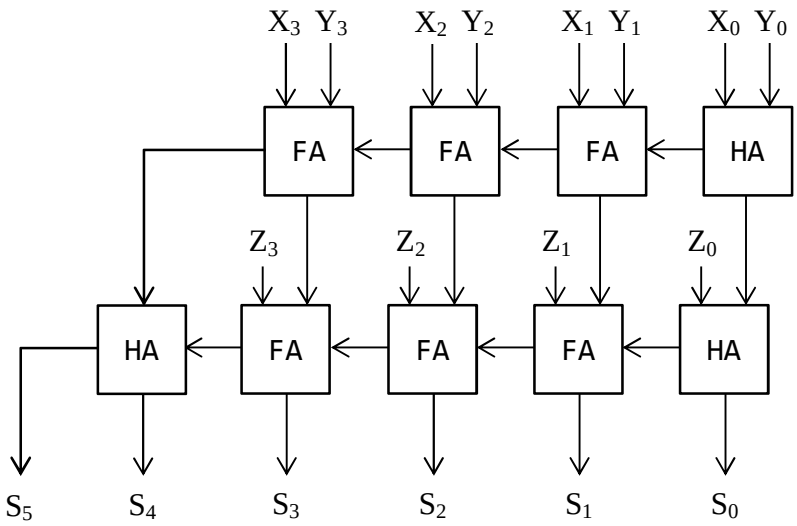
c) Suppose that **X**, **Y**, and **Z** are 4-bit unsigned integers. Let **S = X+Y+Z** be the unsigned sum.

1. What is the number of bits required to represent the output **S**? **(1 Mark)**

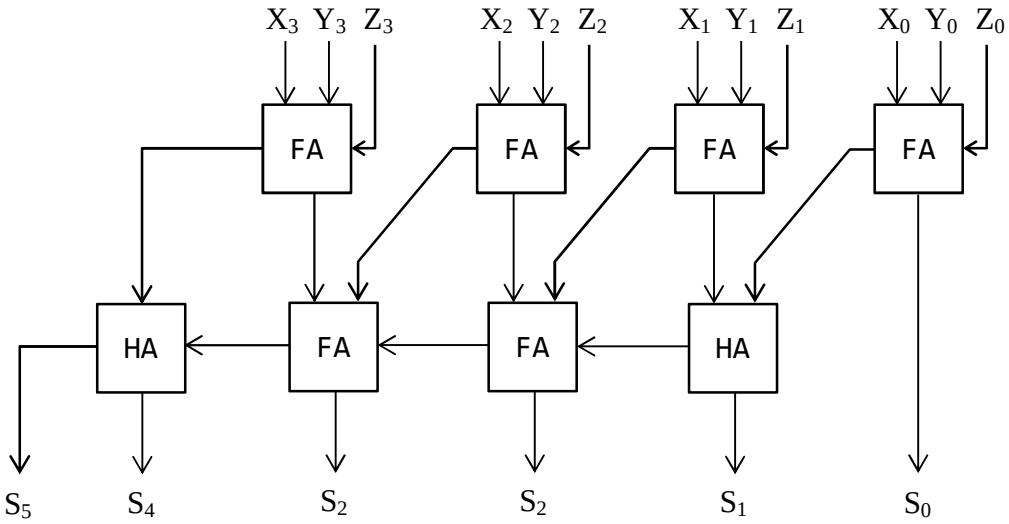
S should have 6 bits to represent the largest sum: 15+15+15 = 45 in binary

2. Draw a circuit that computes **S = X+Y+Z** using only full adders and half adders. **(5 Marks)**

Solution:

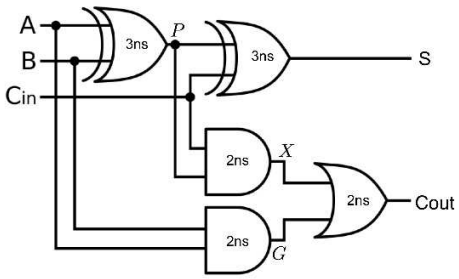


Better Solution:



Question 4 **(13 Marks)**

Consider the following implementation of a full-adder circuit:



a) Write **a gate-level** Verilog description of the above full adder using the Inputs, Outputs, and wire names and gate delays shown on the diagram above. **(5 Marks)**

```
module full_adder(output S, Cout, input A, B, Cin);
     wire P, X, G;

     xor   #3 (P, A, B);      // or assign #3 P = A^B
     xor   #3 (S, P, Cin);         // or assign #3 S = P^Cin
     and   #2 (X, P, Cin);         // or assign #2 X = P & Cin
     and   #2 (G, A, B);      // or assign #2 G = A & B
     or    #2 (Cout, X, G);   // or assign #3 Cout = X | G

endmodule
```

b) Write another Verilog description of the full adder, with delay modeling, using **a single assign statement per output (i.e. two assign statements).** **(3 Marks)**

```
module full_adder(output S, Cout, input A, B, Cin);

     assign #6 S = A^B^Cin ;
     assign #7 Cout = (A^B)& Cin | A & B;

endmodule
```

if someone does assign #6 {Cout,S} = A + B + Cin ; he loses 1 mark for wrong delay modeling of Ci and even if he uses #7

c) Using the full adder in (a) above, write a testbench to test **a 2-bit adder** (i.e. uses two of the full adder in (a)) to add two 2-bit numbers **A** and **B** with a carry in **Ci**, and produces a 2-bit Sum **S** and carry out **Co**.

The testbench should test the 2-bit adder for the following input values (with 10 time units delay in-between the two input patterns): **(5 Marks)**

A = 00, B = 00, Cin = 0, and the second input is **A = 10, B = 01, Cin = 1**.

```
module tb;

     reg [1:0] A, B;
     reg Ci;
     wire [1:0] S, Co;

     full_adder fa1(S[0], Co[0], A[0], B[0], Ci);
     full_adder fa2(S[1], Co[1], A[0], B[0], Co[0]);

     initial begin
           A[0]= 0 ; A[1]= 0 ;
           B[0]= 0 ; B[1]= 0 ;
           Ci= 0 ;
      #10  A[0]= 0 ; A[1]= 1 ;
           B[0]= 1 ; B[1]= 0 ;
           Ci= 1 ;
     end

endmodule
```

OR

```
module tb;

     reg A1,A0, B1,B0;
     reg Ci;
     wire S1, S0, Co0, Co1;

     full_adder fa1(S0, Co0, A0, B0, Ci);
     full_adder fa2(S1, Co1, A1, B1, Co0);

     initial begin
           A0= 0 ; A1= 0 ;
           B0= 0 ; B1= 0 ;
           Ci= 0 ;
      #10  A0= 0 ; A1= 1 ;
           B0= 1 ; B1= 0 ;
           Ci= 1 ;
     end

endmodule
```