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 142 (Spring 2015)

Major Exam II

Saturday April 18, 2015

Time: 150 minutes, Total Pages: 11

Name:_KEY_____________________________ 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

Question	Maximum Points	Your Points
1	12	
2	12	
3	12	
4	12	
5	10	
6	10	
Total	68	

Question 1. **(12 Points)**

Assuming the availability of all variables and their complements, simplify the following two Boolean functions F and G subject to the given don't care conditions d1 and d2 using the K-Map method:

(i) Implement F using **only** **NOR** gates:

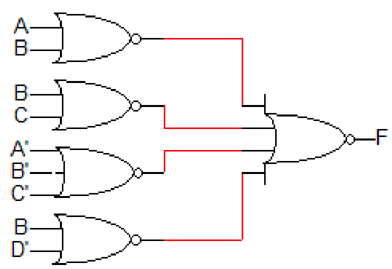
$F(A, B, C, D) = \sum(4, 5, 6, 10, 12, 13)$

$d1(A, B, C, D) = \sum(3, 7, 9)$

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

To get a 2-Level NOR-NOR implementation, we use the simplified POS expression (Groups of 0's) given by:

$F = (A+B) \cdot (B+C) \cdot (A'+B'+C') \cdot \{(C'+D') \textbf{ or } (B+D')\}$



(ii) Implement G using **only** **NAND** gates:

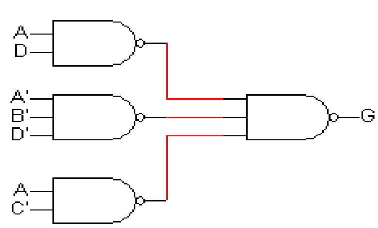
$G(A, B, C, D) = \sum(0, 2, 8, 11, 13, 15)$

$d2(A, B, C, D) = \sum(3, 6, 7, 9, 12)$

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

Simplified SOP expression directly maps into a 2-Level NAND-NAND implementation.

$G = A D + A' B' D' + A C'$ OR $G = A D + B' C' D' + A' C$



Question 2. **(12 Points)**

Design a combinational logic circuit which receives a **4-bit** unsigned number ***X*** **($x_3\ x_2\ x_1\ x_0$)** as input and produces an output ***Z*** which equals the result of integer division of ***X*** **by 3 (**e.g., if X=7, Z=2).

(i) How many bits does the output ***Z*** have? Why? (2 Points)

Max output value = 15/3 = 5 → 3-bits

(ii) Derive the truth table of this circuit. (4 Points)

x_3	x_2	x_1	x_0	Z_2	Z_1	Z_0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	0	0	0
0	0	1	1	0	0	1
0	1	0	0	0	0	1
0	1	0	1	0	0	1
0	1	1	0	0	1	0
0	1	1	1	0	1	0
1	0	0	0	0	1	0
1	0	0	1	0	1	1
1	0	1	0	0	1	1
1	0	1	1	0	1	1
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	1	0	1

$x_3\ x_2$ \ $x_1\ x_0$	00	01	11	10
00			1	
01	1	1		
11			1	
10		1	1	1

$Z_0 = \overline{x_3}x_2\overline{x_1} + x_1x_0(x_3 + \overline{x_2}) + x_3\overline{x_2}(x_1 + x_0)$

(iii) Using K-maps, derive minimized sum-of-products expression(s) for the circuit output(s).

(6 points)

$Z_0 = \overline{x_3}x_2\overline{x_1} + x_1x_0(x_3 + \overline{x_2}) + x_3\overline{x_2}(x_1 + x_0)$

$Z_1 = x_3\overline{x_2} + \overline{x_3}x_2x_1$

$Z_2 = x_3x_2$

Question 3. **(12 points)**

(i) Fill the following table with the appropriate signed number representation. Under the columns labeled "O" put "T" if there is an overflow, otherwise put "F". If the value cannot be represented correctly using the specified number of bits, put "NA". (6 points)

# Bits	Sign-Magnitude	O	1's Complement	O	2's Complement	O	Decimal Value
5	NA	T	NA	T	10000	F	-16
7	011 1111	F	0111111	F	0111111	F	63
8	10001100	F	1111 0011	F	1111 0100	F	-12
6	11 0001	F	10 1110	F	10 1111	F	-17

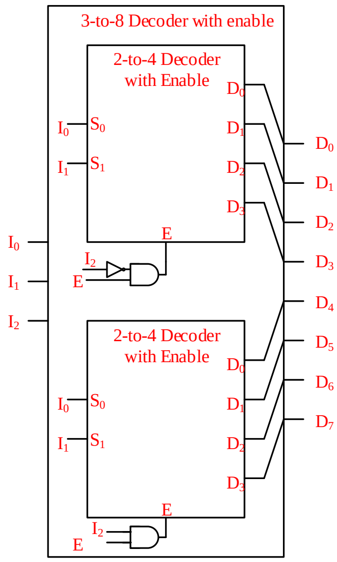
(ii) Perform the following signed-2's complement arithmetic operations in binary using 5 bits. All numbers given are represented in the signed-2's complement notation. <u>Indicate clearly the carry values from the last two stages</u>. For each of the three operations, check and indicate whether overflow occurred or not. (6 points)

11000 -10010 11 11000 +01110 00110	01001 -11001 01 01001 +00111 10000	11011 +10011 10 11011 +10011 01110
Overflow? (No)	Overflow? (Yes)	Overflow? (Yes)

Question 4. **(12 Points)**

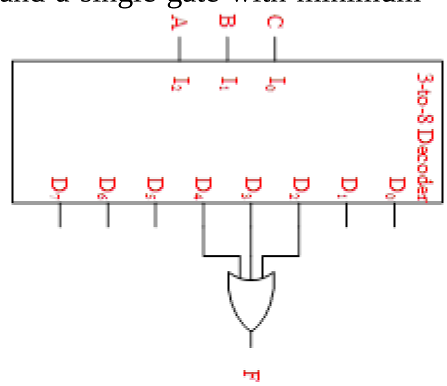
In the following questions, you must clearly label all inputs/outputs of all MSI components, and clearly indicate both the MSB and LSB.

(i) Implement a 3-to-8 decoder with enable, using two 2-to-4 decoders with enable and other logic gates as needed. (5 points)

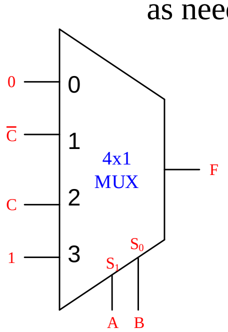


(ii) Impalement $F(A,B,C)=M_0.M_1.M_5.M_6.M_7$ using a decoder and a single gate with minimum number of inputs. (3 points)

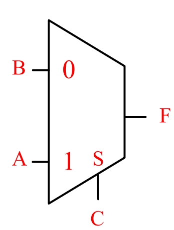
$F(A,B,C)=m_2+m_3+m_4$



(iii) Implement $F(A,B,C)= m_2+m_5+m_6+m_7$, using the smallest possible multiplexer and inverters as needed. (4 points)



OR using k-map F=C'B+CA



A \ BC	00	01	11	10
0	0	0	0	1
1	0	1	1	1

Question 5. **(10 Points)**

Given two 8-bit signed numbers, **X** and **Y** in *2's complement* representation, and assuming overflow does not occur:

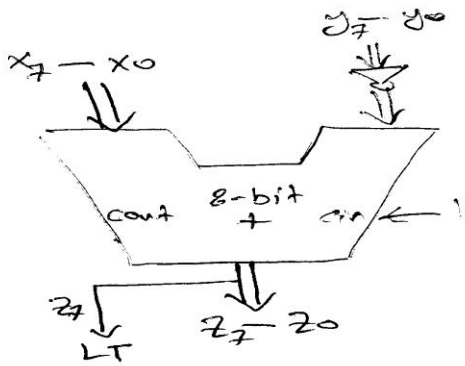
(i) Using a *single* adder of any size and basic logic gates, design a circuit that generates a signal **LT** which equals **1** if **X < Y**, otherwise it is **0**. (2 Points)

(ii) Using a *single* adder of any size and basic logic gates, design a circuit that receives an 8-bit signed number **M** and produces an output value which equals **3*M**. (2 Points)

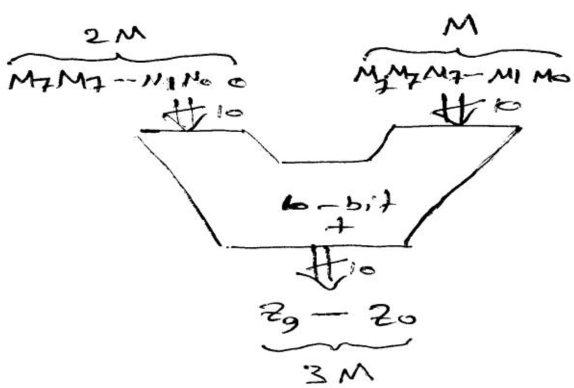
(iii) Given two 8-bit signed numbers **X** and **Y** in *2's complement* representation, use only adders of any size, multiplexers and basic logic gates, to compute the output ***Z*** defined as follows: (6 Points)

IF $(X \geq Y)$ **Then** $Z = 3*(X - Y)$
Else $Z = 2*(Y - X)$

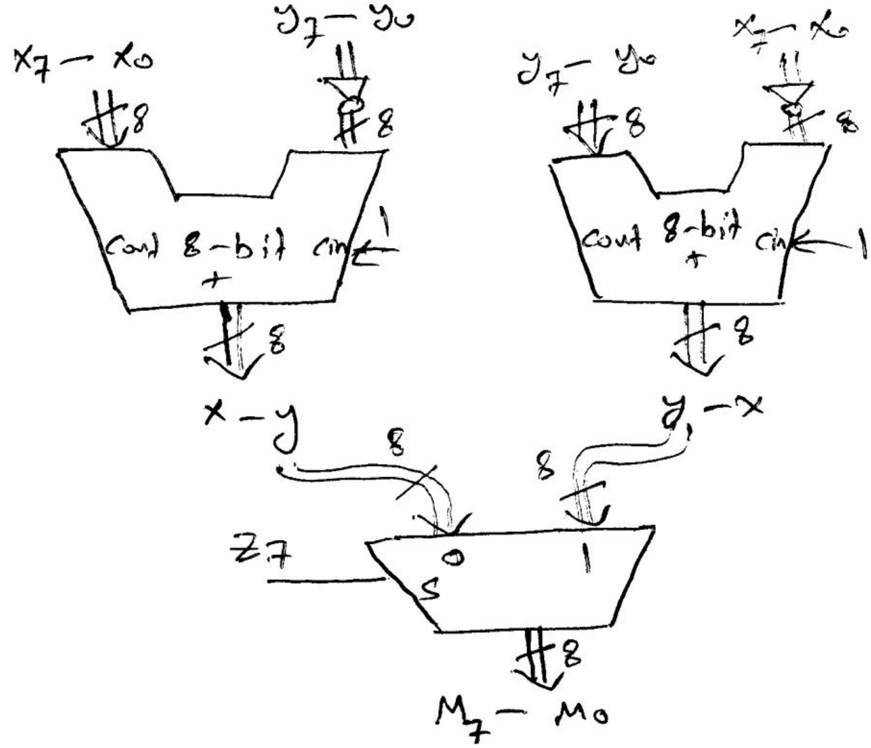
(i)

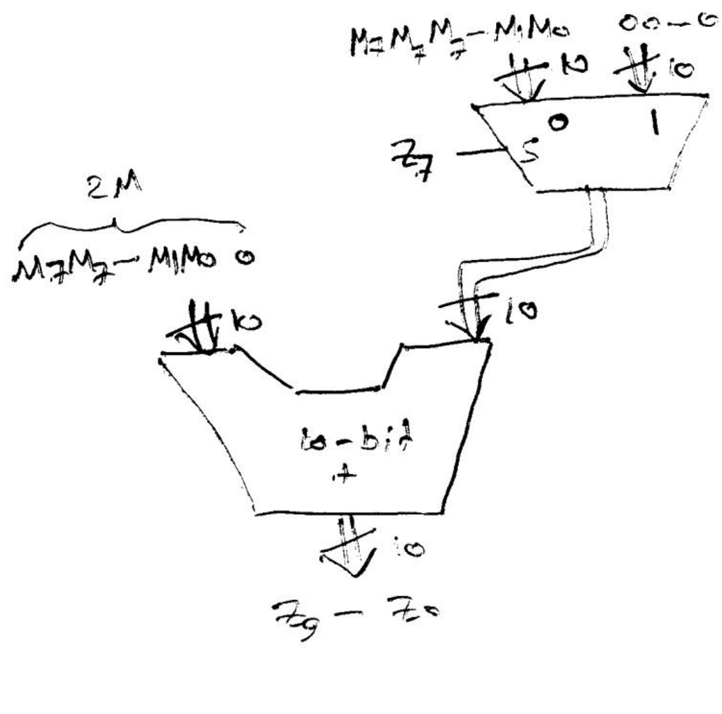


(ii)



(iii)


x7 — x0
y7 — y0
8
cout 8-bit cin
+
1
x - y
y - x
Z7
0
1
S
M7 — M0

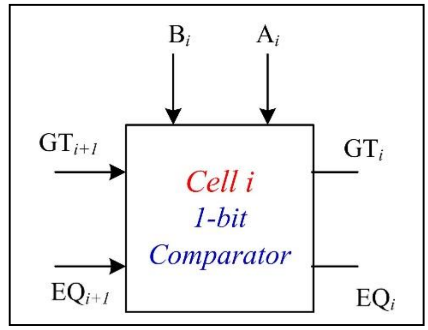

2M
10
0
1
S
Z7
10-bit
+
Z9 — Z0

Question 6. **(10 Points)**

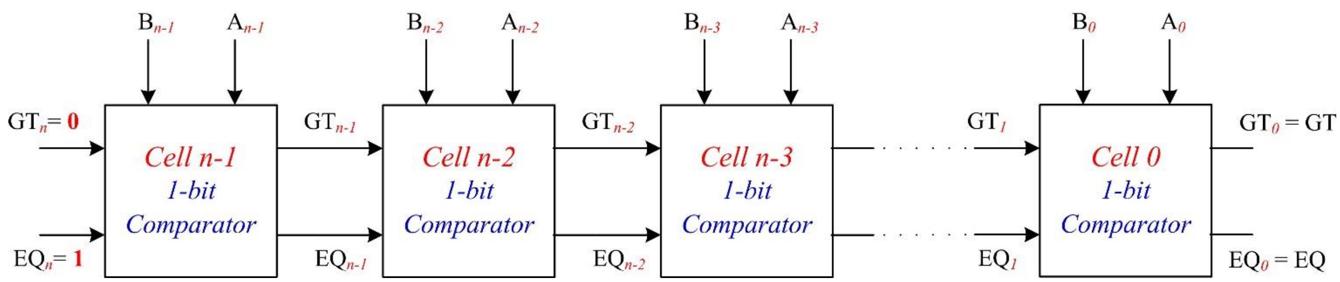
	GT	EQ
IF A > B	1	0
IF A = B	0	1
IF A < B	0	0

Given below the design of an *n*-bit magnitude comparator. The circuit receives two *n*-bit unsigned numbers ***A and B*** and produces two outputs **GT** and **EQ** as given in the table to the right.

The input operands are processed in a bitwise manner *starting with the most significant bit (MSB)*. The comparator circuit is constructed using *n identical copies* of the basic 1-bit *cell* shown to the right.



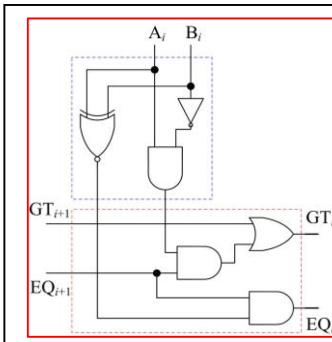
The Figure below shows the *n*-bit comparator circuit implemented using *n* copies of the basic 1-bit cell.



Boolean expressions of the outputs of ***cell i*** and its gate-level implementation are given below:

$$GT_i = GT_{i+1} + A_i\,\bar{B}_i\,EQ_{i+1}$$

$$EQ_i = (A_i \odot B_i).\,EQ_{i+1}$$



Cell *i* Gate Level Implementation

(i) Write a Verilog model **Comp1Bit** to model the 1-bit comparator circuit using *either* a structural model of basic logic gates *or* a behavioral model using the **assign** statement.

(4 Points)

The declaration of the Comp1Bit module is as follows:

```
module Comp1Bit (output GT_out, EQ_out ,
                 input  GT_in  , EQ_in, Ai, Bi);
```

```
module Comp1Bit (output GT_out, EQ_out, input GT_in, EQ_in, Ai, Bi);

        assign GT_out = GT_in || Ai && !Bi && EQ_in;
        assign EQ_out = (Ai ~^ Bi) && EQ_in;

endmodule
```

OR

```
module Comp1Bit (output GT_out, EQ_out, input GT_in, EQ_in, Ai, Bi);

not (g1, Bi);
and (g2, g1, Ai);
xnor(g3, Ai, Bi);
and (g4, g2, EQ_in);
or  (GT_out, GT_in, g4);
and (EQ_out,  EQ_in, g3);

endmodule
```

(ii) Complete the following Verilog model **Comp3Bit** which models a 3-bit comparator circuit.

(2 Points)

```
module Comp3Bit (output Greater, Equal,
                        input [2:0] A , B) ;

wire  [2:1]  GT, EQ ; // internal wires connecting cells

/* First instance "M1" of the cell Comp1Bit with its inputs GT_in and
EQ_in connected to fixed values of 0 and 1 respectively */
//
Comp1Bit M1 (GT[2] , EQ [2], 1'b0, 1'b1, A[2], B[2]);
Comp1Bit M2 (GT[1], EQ[1], GT[2], EQ[2], A[1], B[1]);
Comp1Bit M3 (Greater, Equal, GT[1], EQ[1], A[0], B[0]);

endmodule
```

(iii) Write a Verilog test bench to test the 3-bit comparator **Comp3Bit** by applying the following input patterns consecutively with a delay of 20ps: (4 Points)

1. {A=100, B=011},
2. {A=101, B=101},
3. {A=011, B=111}.

```
module t_Comp3Bit();

  wire   Greater, Equal;
  reg     [2:0] A, B;

  Comp3Bit M1 (Greater, Equal, A, B) ;


initial begin
    A=3'b100; B=3'b011;
    #20 A=3'b101; B=3'b101;
    #20 A=3'b011; B=3'b111;
  end
endmodule
```

Verilog Primitives

- Basic logic gates only
  - and
  - or
  - not
  - buf
  - xor
  - nand
  - nor
  - xnor

These gates are expandable: 1st node is O/P node, followed by 1, 2, 3 … number of input nodes

Verilog Operators

{ }	concatenation
+ - * / **	arithmetic
%	modulus
> >= < <=	relational
!	logical NOT
&&	logical AND
||	logical OR
==	logical equality
!=	logical inequality
===	case equality
!==	case inequality
? :	conditional

~	bit-wise NOT
&	bit-wise AND
|	bit-wise OR
^	bit-wise XOR
^~ ~^	bit-wise XNOR
&	reduction AND
|	reduction OR
~&	reduction NAND
~|	reduction NOR
^	reduction XOR
~^ ^~	reduction XNOR
<<	shift left
>>	shift right