BILKENT UNIVERSITY
Department of Electrical and Electronics Engineering
EEE102 Introduction to Digital Circuit Design
Midterm Exam Solution

6-04-2007

Duration 120 minutes

Surname: _____

Name: _____

ID-Number: _____

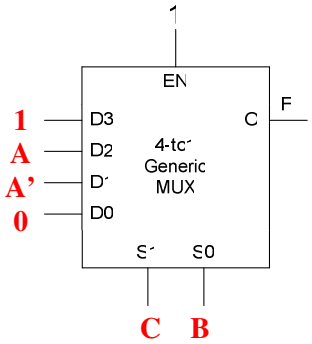
Signature: _____

There are 5 questions of equal weight. Solve all.
Do not detach pages. Show all your work.

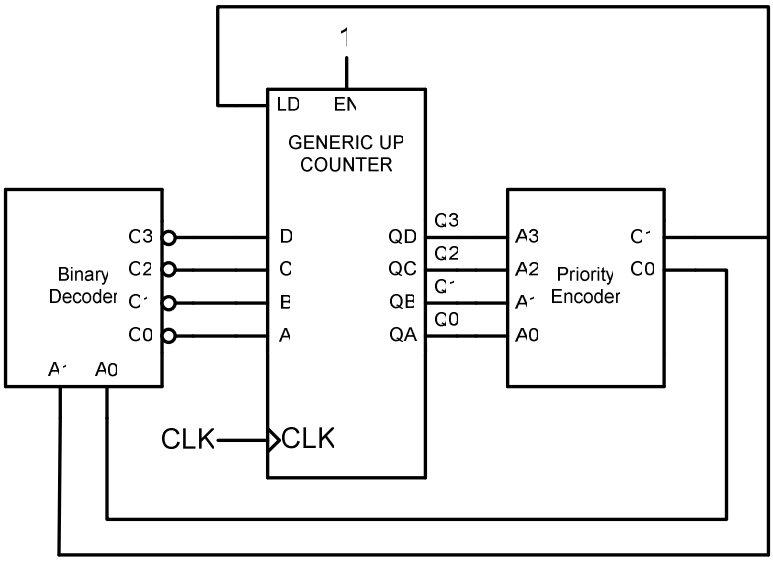
Q1	
Q2	
Q3	
Q4	
Q5	
Total	

Q1. a) (4 points) Implement the function F=A'B+AC by using the following 4-to-1 generic multiplexer. You may assume that the inverted inputs are available. No other logic gates are allowed.

One possible solution is as the following. There are other ways too.



b) (16 points) Determine the counting sequence of the following configuration and fill in the following table. Assume that the counter is initially 0. Priority order is A3, A2, A1, and A0 from high to low in the priority encoder.



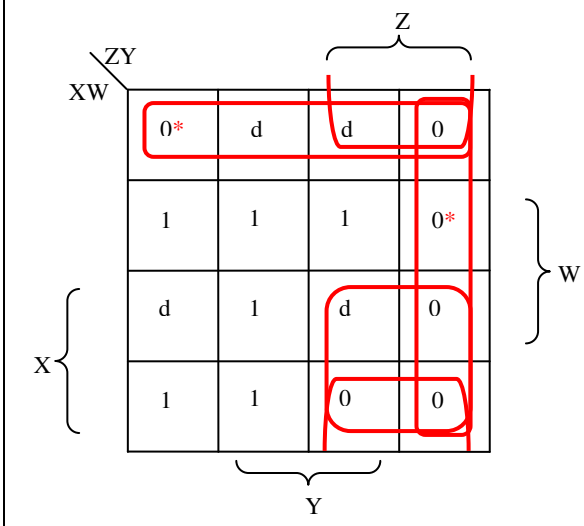
Next State Table to determine the counting sequence:

Q3	Q2	Q1	Q0	LD	Q3*	Q2*	Q1*	Q0*
0	0	0	0	0	0	0	0	1
0	0	0	1	0	0	0	1	0
0	0	1	0	0	0	0	1	1
0	0	1	1	0	0	1	0	0
0	1	0	0	1	1	0	1	1
1	0	1	1	1	0	1	1	1
0	1	1	1	1	1	0	1	1
1	0	1	1	1	0	1	1	1

Counting sequence: 7 – 11 – 7 – 11 – 7 – 11…..

Q2. a) (5 points) Find all minimized POS expressions for the following TT.

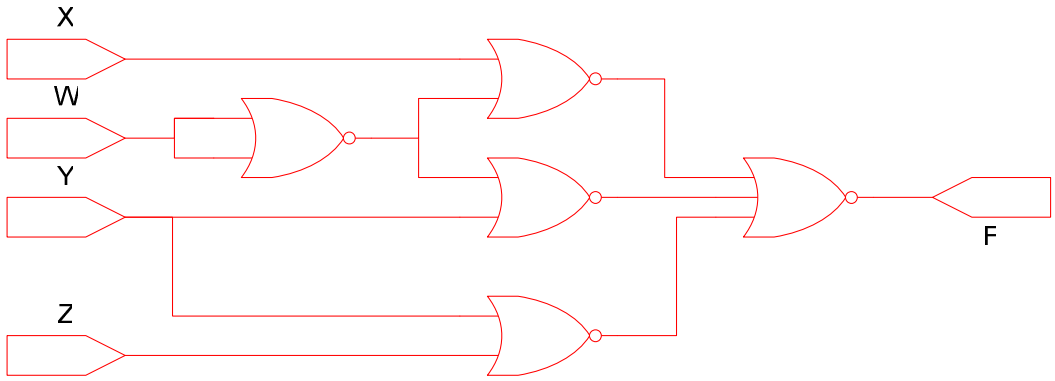
Z	Y	X	W	F
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	d
0	1	0	0	d
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	d
1	1	0	1	1
1	1	1	0	0
1	1	1	1	d



F1 = (X+W)(Z'+Y)(Z'+W) F2 = (X+W)(Z'+Y)(Z'+X')

b) (5 points) Draw the schematics of one of the above minimized functions by using NOR gates only.

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



c) (5 points) Expand F=(X+Y').Z + (X'.Y.Z') using Shannon's expansion theorem with respect to Y.

Shannon's expansion theorem is as follows:

F(X1, X2, …. Xn) = X1.F(1,X2,,…..Xn)+ X1'.F(0,X2,,…..Xn)

F = Y(XZ+X'Z') + Y'(Z)

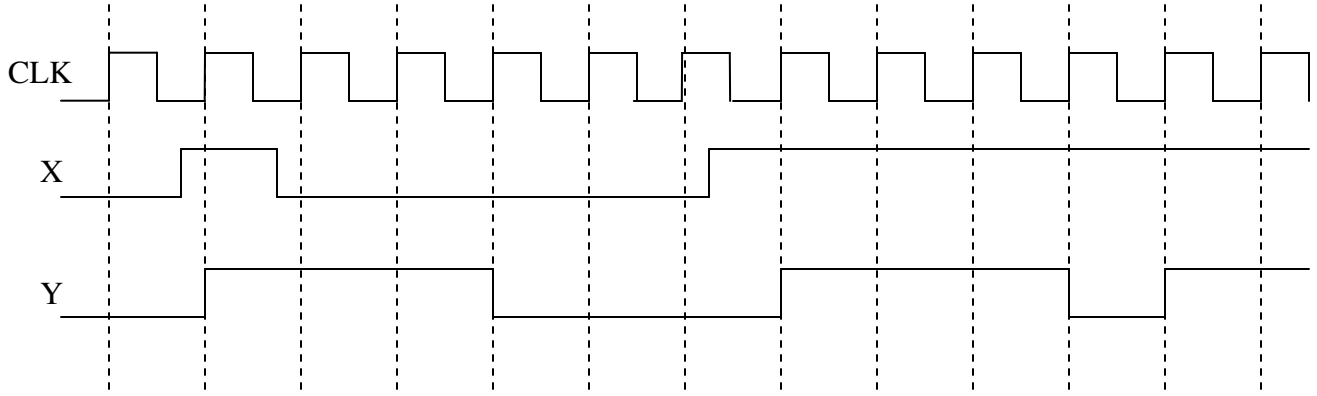
d) (5 points) Prove using algebraic manipulation that

(X+Y').Z + (X'.Y.Z') = (X+Y'+Z').(X'+Z).(Y+Z)

Duality: (XY'+Z)(X'+Y+Z')=(XY'Z')+(X'Z)+(YZ)

Multiply Out Left Side: XY'Z'+X'Z+YZ (Note that XX'=YY'=ZZ'=0)

Q3. A synchronous FSM has one input X and one output Y. Y is 0 initially (at Power ON). If at a clock +ve tick X = 1 then Y becomes 1 for three clock periods and then returns to 0. If X is 1 during this time then it is not effective. Write VHDL code to describe this circuit. The timing diagram below is given for you to understand the problem better.



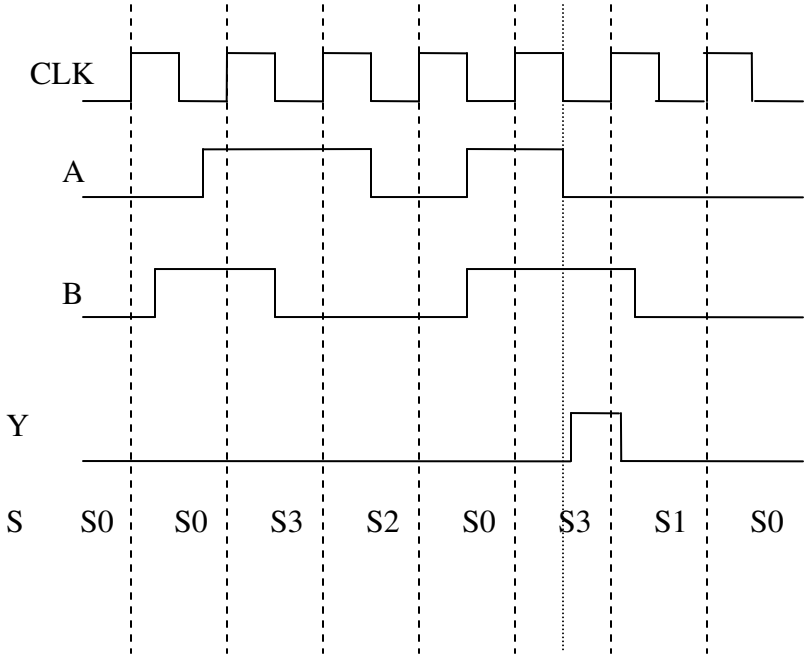
Solution:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity OneShot is
   Port ( clock : in std_logic;
          X : in std_logic;
          Y : inout std_logic);
end OneShot;

architecture Behavioral of OneShot is
signal flag:std_logic :='0';
begin
Y<=flag;
process(clock)
variable count:std_logic_vector (1 downto 0) :="00";
begin
        if clock'event and clock='1' then
                if flag='0' then
                        if X='1'then flag<='1';end if;
                elsif flag ='1' then
                        count:=count+1;
                        if count="11" then flag<='0';count:="00";end if;
                end if;
        end if;
end process;

end Behavioral;
```

Q4. The VHDL code of an FSM is given below.

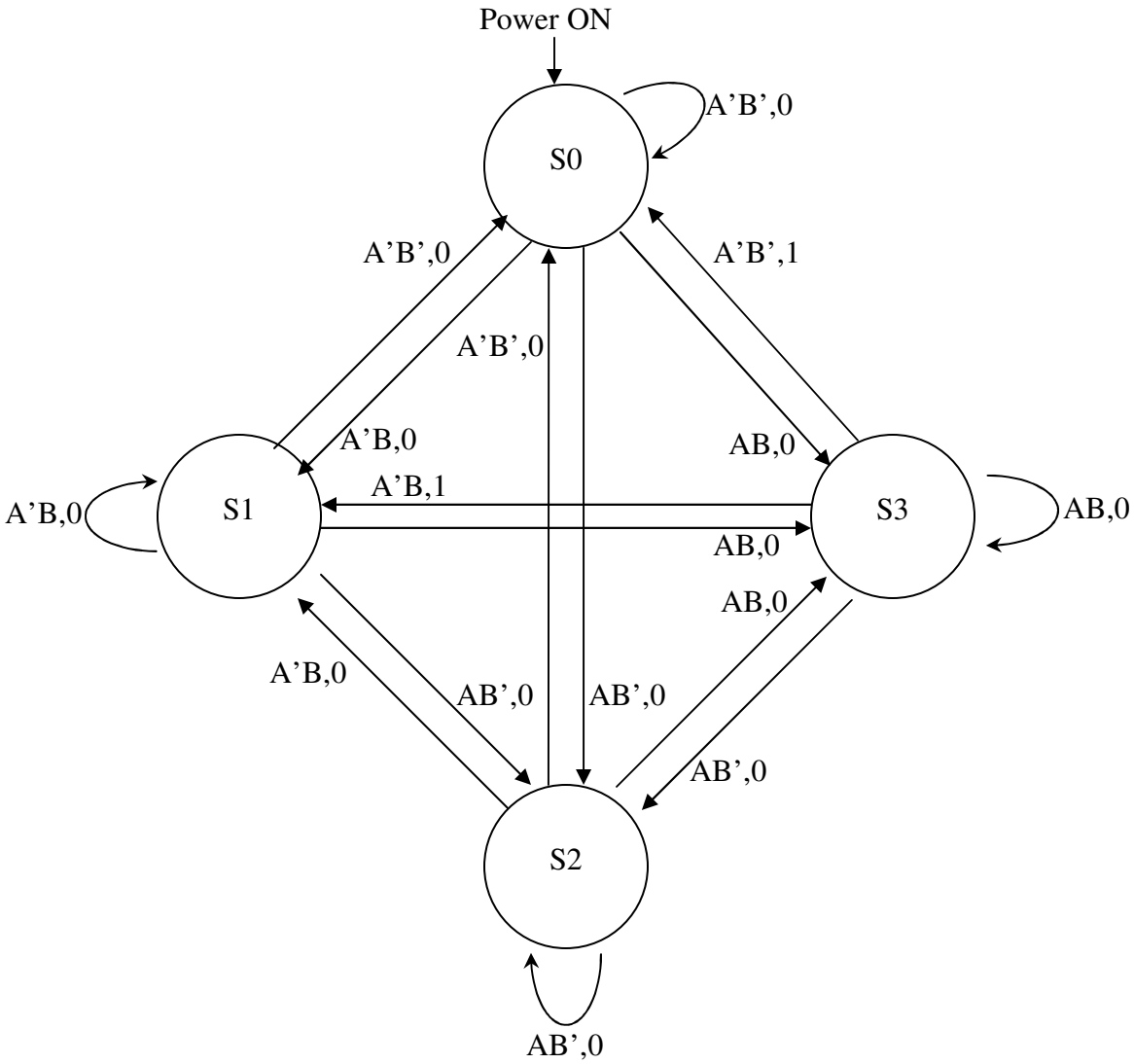
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity TwoInputFSM is
   Port ( CLK : in std_logic;
          A : in std_logic;
          B : in std_logic;
          Y : out std_logic);
end TwoInputFSM;

architecture Behavioral of TwoInputFSM is
type newtype is (S0,S1,S2,S3);
signal S:newtype :=S0;
signal N:newtype;

begin
S<=N when CLK'event and CLK='1' else S;
N<=S1 when (not A and B)='1' else S2 when (A and not B)='1' else S3 when (A and
B)='1' else S0 when (not A and not B)='1';
Y<='1' when S=S3 and A='0' else '0';
end Behavioral;
```

a) Draw the waveform of Y for the given waveforms for A and B.
Solution:

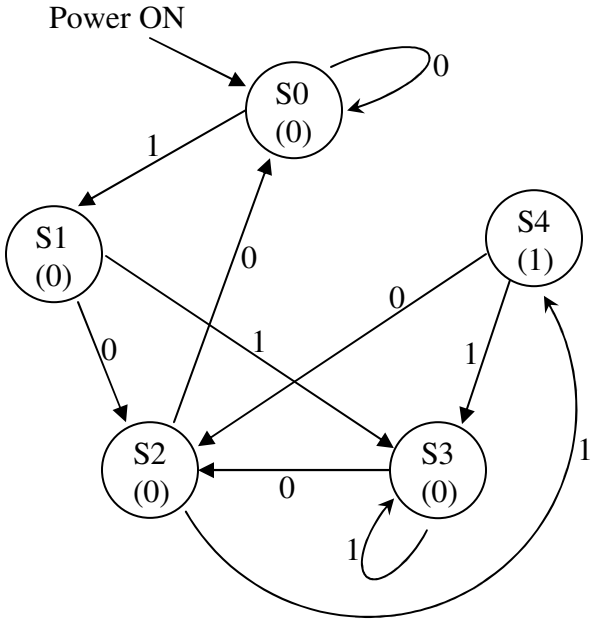


b) **Draw the state/output diagram of this FSM**

Solution:



Q5. Design and draw the FSM which has the state/output diagram shown below. Show all your design steps. What does this FSM do?



Considering that S1 and S3 are equivalent states,
State Encoding:

State	Q1	Q0
S0	0	0
S1	0	1
S2	1	0
S4	1	1

Next State Table:

Q1	Q0	X	Q1*	Q0*
0	0	0	0	0
0	0	1	0	1
0	1	0	1	0
0	1	1	0	1
1	0	0	0	0
1	0	1	1	1
1	1	0	1	0
1	1	1	0	1

Excitation Table: Same as NS table.

Next State Logic:

Q1*=D1=Q0X'+Q1Q0X

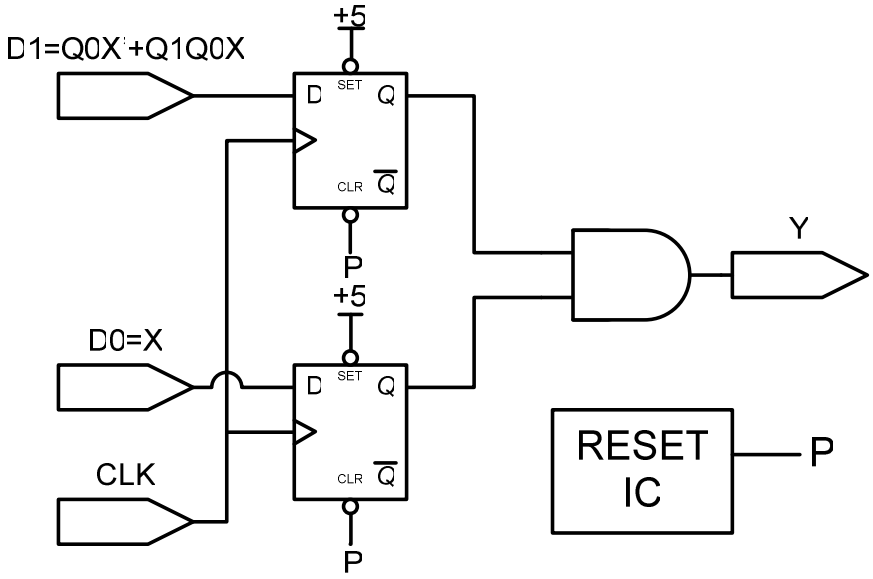
Q0*=D0=X

Output Table:

Q1	Q0	Y
0	0	0
0	1	0
1	0	0
1	1	1

Y=Q1.Q0

Circuit:



Function:

Detects "101" sequence from the input.

FALL 2009: ***2-11-2009***

BILKENT UNIVERSITY

Department of Electrical and Electronics Engineering

EEE102 Introduction to Digital Circuit Design

Midterm Exam I - SOLUTIONS

Surname: _____

Name: _____

ID-Number: _____

Signature: _____

Duration is 120 minutes. Solve all 7 questions. Show all your work.
No books, notes, or calculators.

Q1 (15 points)	
Q2 (15 points)	
Q3 (10 points)	
Q4 (18 points)	
Q5 (15 points)	
Q6 (12 points)	
Q7 (15 points)	
Total	

Q1. (15 points) (a) Perform subtraction on the given unsigned binary numbers using the 2's complement of the subtrahend. You should show all of your arithmatic operations in binary form. Also, give your final answer in decimal representation.

(i) (1.5 pts) 10011 – 10001 = (+2)$_{10}$

```
 1111110  (carries)      10001 → (2's complement) 101111
   010011
   101111
 +-------------
 1  000010
```

(ii) (1.5 pts) 100010 – 100011 = (-1)$_{10}$

```
 00000000  (carries)     0100011 → (2's complement) 1011101
   0100010
   1011101
 +-------------
   1111111               1111111 → (2's complement) 0000001
```

(iii) (1.5 pts) 1001 – 101000 = (-31)$_{10}$

```
 00110000  (carries)     0101000 → (2's complement) 1011000
   0001001
   1011000
 +-------------
   1100001               0011110 → (2's complement) 0011111
```

(b) Perform the following conversions:

(i) (1.5 pts) 4675542_8 = (137B62)$_{16}$ = (0001 0011 0111 1011 0110 0010)$_2$

(ii) (1.5 pts) $1F08B725_{16}$ = (3702133445)$_8$ = (000 011 111 000 010 001 011 011 100 100 101)$_2$

(iii) (1.5 pts) 43.125_{10} = (101011.001)$_2$

	Whole	Fraction
0.125 x 2 =	0 (MSB)	0.25
0.25 x 2 =	0	0.5
0.5 x 2 =	1 (LSB)	0

(c) (6 pts) Convert the BCD number $100010010101.001101110101_{10}$ to binary and (ordinary) decimal form (Note that the subscript $_{10}$ indicates BCD).

Decimal Form = 895.375 (Whole Part of the number found using repeated divisions by 2, and using the remainder as LSB to MSB)

Binary Form = 1101111111.011

	Whole	Fraction
0.375 x 2 =	0 (MSB)	0.75
0.75 x 2 =	1	0.5
0.5 x 2 =	1 (LSB)	0

Q2. (15 points) (a) Simplify the following Boolean expressions to the indicated number of literals. You need to state the name of the Theorems used in in every step of the simplification process. (Note: x+x'yz is a 4 literal expression, since x and x' are counted as 2 literals):

(i) (3 pts) $A'B(D'+C'D)+B(A+A'CD)$ **to one literal**

= A'BD'+A'BC'D+AB+A'BCD (Distributivity) = A'BD'+AB+A'BC'D+A'BCD (Commutativity) = A'BD'+AB+A'BD (Combining) = A'B + AB (Combining) = B (Combining)

(ii) (3 pts) $(x'y'+z)'+z+xy+wz$ **to three literals**

= (x+y).z' + z + xy + wz (DeMorgan's) = xz' + yz'+ xy + z + wz (Distributivity& Associativity) = xz' + yz' + xy + z (Covering) = xz' + xy + yz' + z (Associativity) = xz' + xy + y + z (Proof Below) = (xz' + z) + (xy + y) (Associativity) = x + z + y (Covering Theorem plus Proof below for xz'+z = x + z)

Proof of X + X'Y = X + Y

X.1 + X'Y + Y.1 = X.1 + X'Y (Consensus Theorem) = X + X'Y (Identities Theorem)

So, working with the left-hand side, we get:

X.1 + X'Y + Y.1 = X + X'Y + Y (Identities)
= X + Y.(X' + 1) (Distributivity Theorem)
= X + Y (Null Elements)

Therefore X + X'Y = X + Y

(b) (3 pts) Determine the dual of the following Boolean expression, introducing parentheses as needed: $(x_1 + x_2x_3x_4)'$

$f_D = x_1' + x_2'x_3' x_4'$

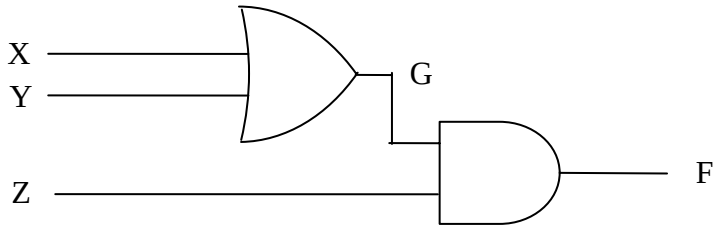
(c) A Boolean function is self-dual if $f(X) = f^d(X)$ **- that is, if the function and its dual are the same. Determine whether the following Boolean expressions are selfdual:**

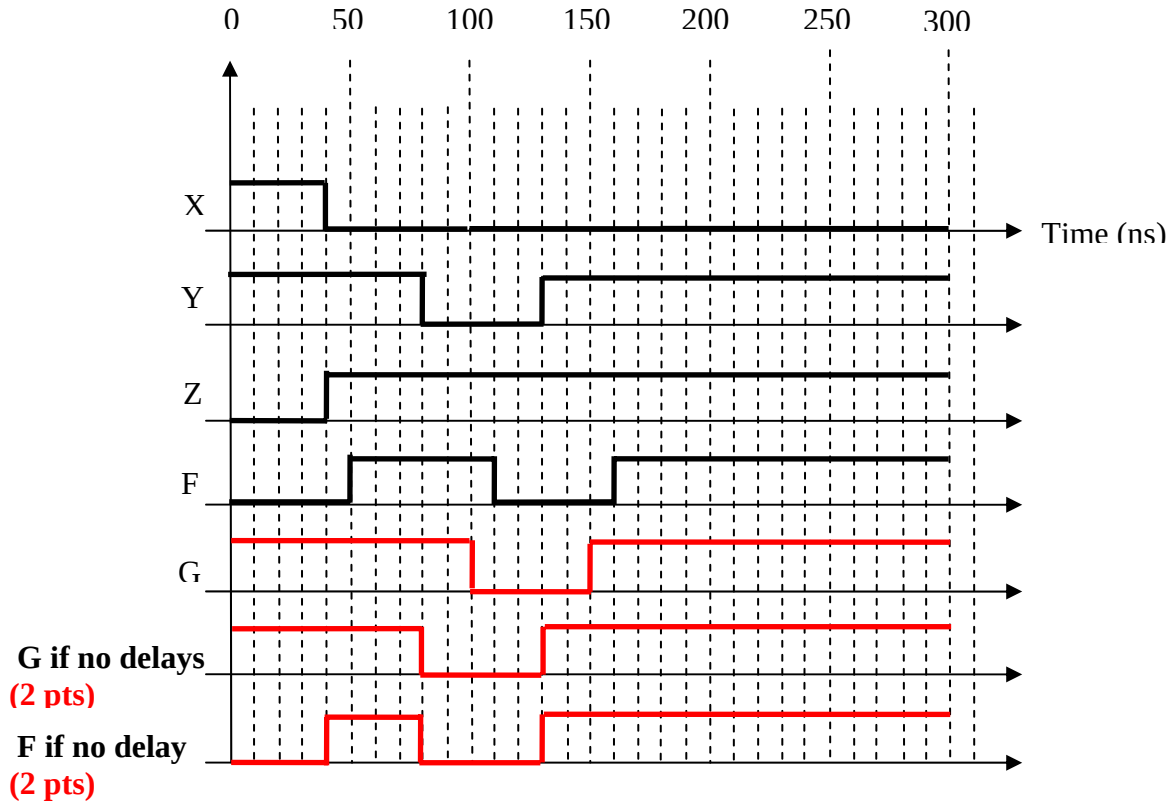
(i) a **(ii)** ab **(iii)** $ab+(ab)'$

Note: *You need to show all of your work for full-credit.*

(i) Self-dual ($f_D = a = f$) (2 pts)
(ii) Not self-dual ($f_D = a+b \neq f$) (2 pts)
(iii) Not self-dual ($f_D = (a+b).(a'+b') \neq f$) (2 pts)

Q3. (10 points) For the circuit given below, a timing simulation is performed and the input and output waveforms are given. The two gates in the circuit have different amounts of delay. Find the delays of each gate and draw G. Explain and justify your answer. Also draw G and F if the gates do not have any delays.





OR Gate = 20 ns (3 pts),
AND Gate = 10 ns (3 pts).

Q4. (18 points) (a) Express the complement of the following functions in sum-of-minterms form:

(i) (4 pts) $F(A,B,C,D)=\sum(3,5,9,11,15)$

$$F'=\sum(0,1,2,4,6,7,8,10,12,13,14)=A'B'C'D'+A'B'C'D+A'BC'D'+A'BCD'+A'BCD+AB'C'D'+AB'CD'+ABC'D'+ABC'D+ABCD'$$

(ii) (4 pts) $F(A,B,C,D)=\prod(2,4,5,7)$

$$F'(A,B,C,D)=\sum(2,4,5,7)=A'B'CD'+A'BC'D'+A'BC'D+A'BCD$$

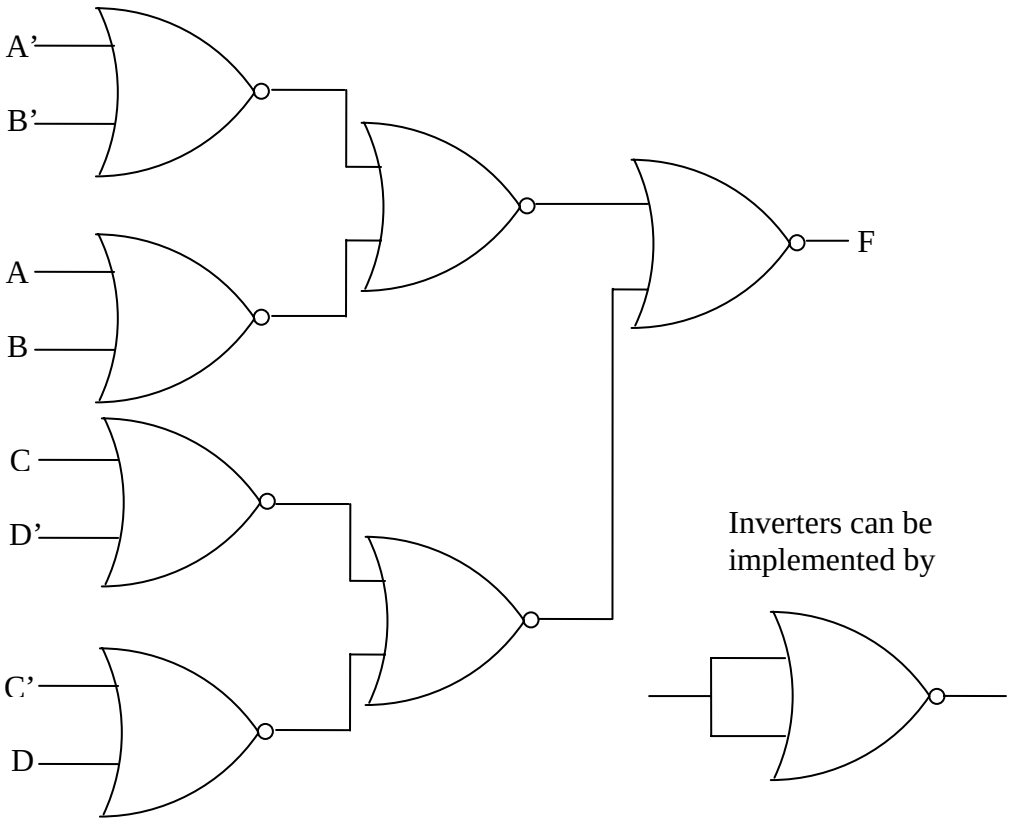
(b) For both parts given below, you should assume that the signals are only available to you in their standard form, and to get their complemented version, you need to implement inverters using NOR gates.

(i) (3 pts) Draw a logic diagram using only two-input NOR gates to implement the following function:

$$F(A,B,C,D)=\overline{(A\oplus B)}(C\oplus D)=(A\oplus B)'(C\oplus D)$$

$$\overline{(A\oplus B)}=AB+A'B'$$

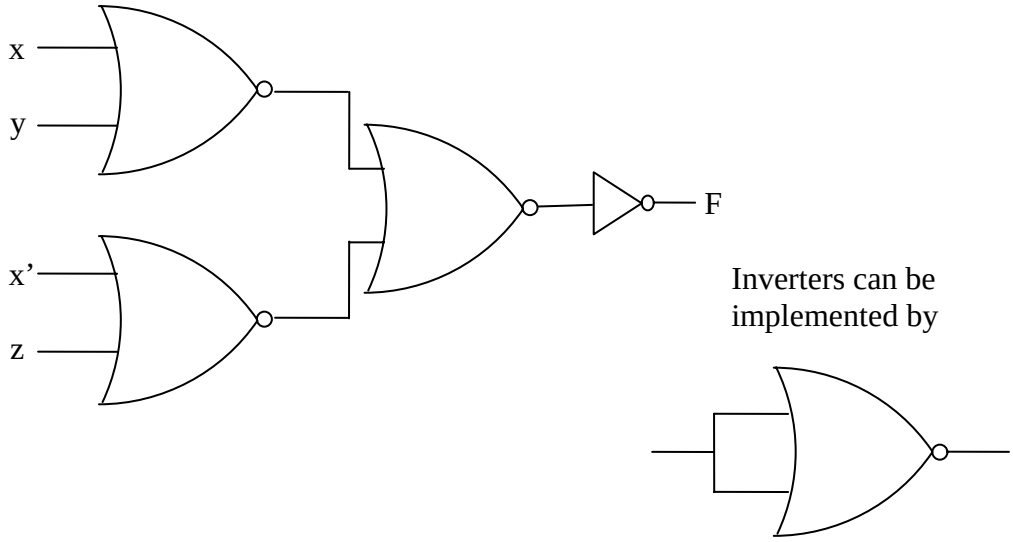
$$(C\oplus D)=C'D+CD'$$



(ii) (3 pts) Implement the following function with two-level NOR gates and draw the circuit. (You may simplify the function before implementation.)

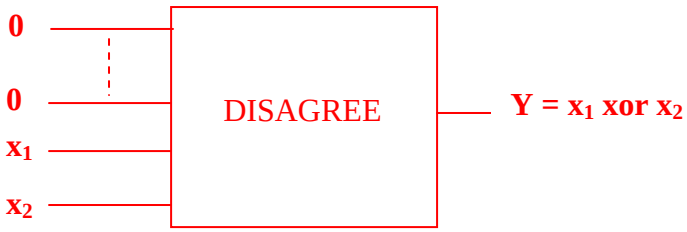
$$F(x, y, z) = \overline{(x + y)(x' + z)} = ((x + y)(x' + z))'$$

F = x'y'+ xz'



(c) (4 pts) We define the DISAGREE function $z(x_1, x_2, ..., x_n)$ to be 1 if and only if $x_i \neq x_j$ for some *i* and *j*. Determine whether or not {DISAGREE} is functionally complete. Prove your answer. (In this context "functionally complete" means "being able to implement any combinational circuit".)

If we can use Disagree to implement NAND or NOR, then Disagree is functionally complete.
No, "Disagree" is not functionally complete, since if we tie all of its inputs but two of them (for example x1 and x2) to '0' (or setting n = 2), then the output is the XOR of the 2 inputs. And, with XOR, we can implement "NOT", but not "AND" or "OR" gates.



Q5. (15 points) The following VHDL code is given for modules "top", "aaa", and "bbb". Draw the circuit schematic of module "top" using "aaa" and "bbb" as components. Find out and explain what "aaa" and "bbb" do and finally state what the function of "top" is.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity top is
   Port ( x : in  STD_LOGIC_VECTOR (2 downto 0);
          y : out  STD_LOGIC_VECTOR (2 downto 0));
end top;

architecture Behavioraltop of top is
signal w:std_logic_vector(2 downto 0);
component aaa
   Port ( a : in  STD_LOGIC;
          b : in  STD_LOGIC;
          c : in  STD_LOGIC;
          z : out  STD_LOGIC_VECTOR (2 downto 0));
end component;
component bbb
   Port ( d : in  STD_LOGIC_VECTOR (2 downto 0);
          z : out  STD_LOGIC_VECTOR (2 downto 0));
end component;
begin
label1: aaa port map(x(2),x(1),x(0),w);
label2: bbb port map(w,y);
end Behavioraltop;


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity aaa is
   Port ( a : in  STD_LOGIC;
          b : in  STD_LOGIC;
          c : in  STD_LOGIC;
          z : out  STD_LOGIC_VECTOR (2 downto 0));
end aaa;

architecture Behaviorala of aaa is
begin
z(2)<= not a;
z(1)<= not b;
z(0)<= not c;
end Behaviorala;
```

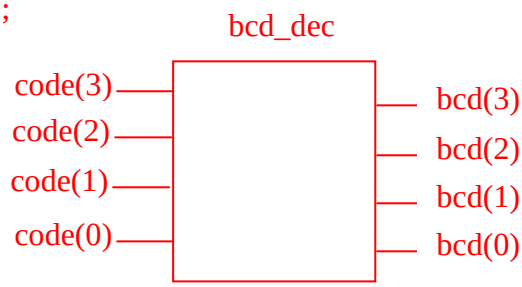
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity bbb is
    Port ( d : in  STD_LOGIC_VECTOR (2 downto 0);
           z : out  STD_LOGIC_VECTOR (2 downto 0));
end bbb;

architecture Behavioralb of bbb is
signal x,y:std_logic_vector(3 downto 0);
begin
y<=x+1;
z(2)<=y(2);
z(1)<=y(1);
z(0)<=y(0);
x(3)<='0';
x(2)<=d(2);
x(1)<=d(1);
x(0)<=d(0);
-- Note that + operator is addition and it works for std logic vectors.
-- However the person who wrote the code is not sure about what happens when '1'
-- is added to "111".
end Behavioralb;
```

(5 pts) aaa: takes the ones complement of a number
(5 pts) bbb: adds 1 to the number
(5 pts) top: finds the 2’s complement for 3-bit 2’s complement numbers

Q6. (12 points) Write a VHDL code using behavioral-type programming to implement an 84-2-1 to BCD code converter. The 84-2-1 code system is shown in the table below. If the input is other than the 84-2-1 codes shown below then the output should be the binary equivalent of the hexadecimal number E (meaning Error).

Decimal Digit	8, 4, -2, -1 Code	BCD Code
0	0000	0000
1	0111	0001
2	0110	0010
3	0101	0011
4	0100	0100
5	1011	0101
6	1010	0110
7	1001	0111
8	1000	1000
9	1111	1001

```
library ieee;
library ieee.std_logic_1164.all;

entity bcd_dec is
        port( code: in std_logic_vector (3 downto 0);
              bcd: out std_logic_vector (3 downto 0) );
end bcd_dec;

architecture bcddecoder of bcd_dec is
        process(code)
        begin
          case CONV_INTEGER(code) is
                when 0 => bcd <= “0000”;
                when 7 => bcd <= “0001”;
                when 6 => bcd <= “0010”;
                when 5 => bcd <= “0011”;
                when 4 => bcd <= “0100”;
                when 11 => bcd <= “0101”;
                when 10 => bcd <= “0110”;
                when 9 => bcd <= “0111”;
                when 8 => bcd <= “1000”;
                when 15 => bcd <= “1001”;

                when others => bcd <= “0111”;
           end case;
        end process;
end bcddecoder;
```



Q7. (15 points) Given the function $Z(A,B,C,D) = \sum(6,7,8,9,14) + d(1,5,11,15)$

(a) (3 pts) Find all minimal sums for F

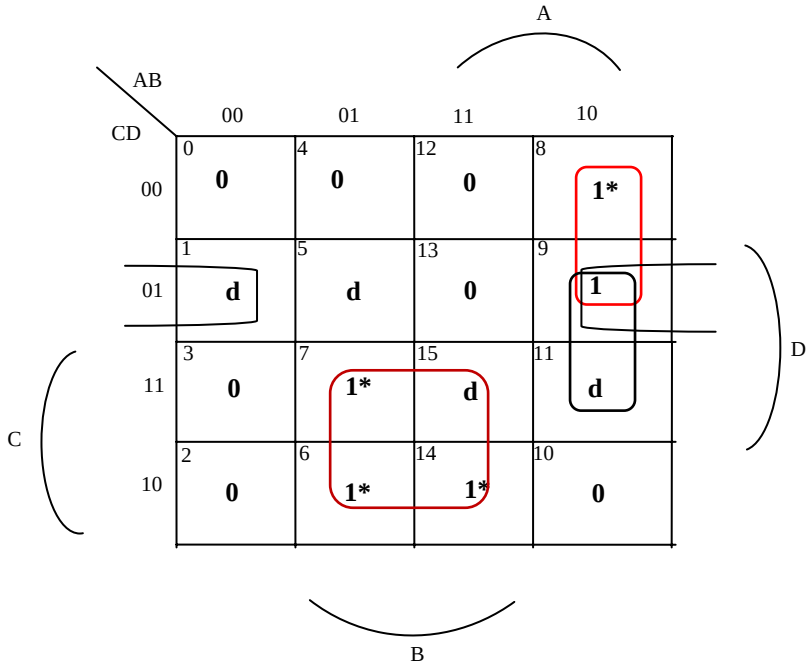
(b) (6 pts) Final all minimal products for F

(c) (3 pts) Which of the above minimal sums are equivalent to which of the above minimal products?

(d) (3 pts) Which of the above found expressions would you use for implementation. Why?

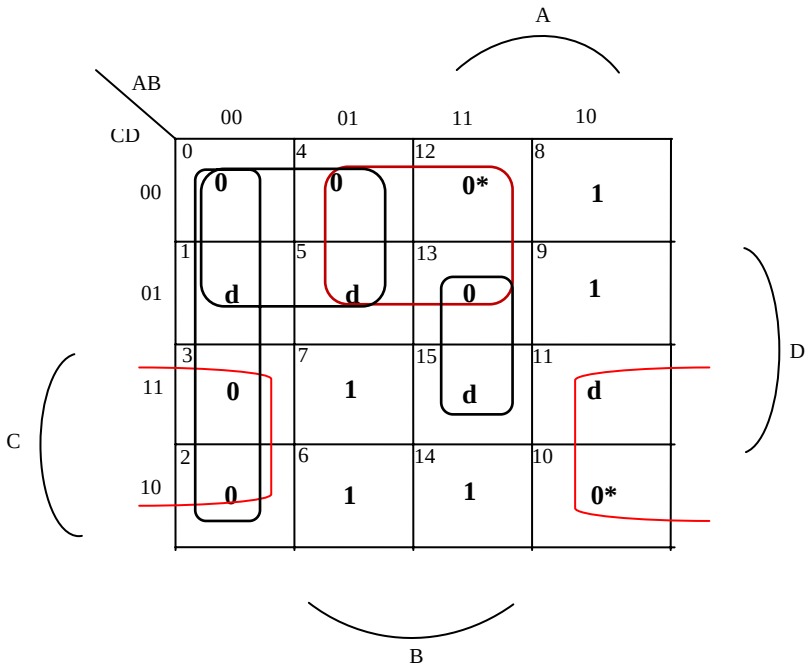
Solution:

(a)



F_1 = BC+AB'C' There is only one minimal sum

(b)



F_2 = (B'+C)(B+C')(A+C)
F_3 = (B'+C)(B+C')(A+B)
There are two minimal products

(c) In all solutions 1,5,11 are zero and 15 is one. Therefore they are all equivalent.
(d) I would use F_1 because it has two terms while the others have three terms.

Appendix: Some VHDL Templates 18-12-2008

ENTITY DECLARATION

```
entity entity_name is
        generic ( constant_names : constant type;
                  constant_names : constant type;
                  …
                  constant_names : constant type);
        port (  signal_names : mode signal_type;
                signal_names : mode signal_type;
                …
                signal_names : mode signal_type);
end entity_name;
```

ARCHITECTURE DEFINITIONS

```
architecture architecture-name of entity-name is
        type declarations
        signal declarations
        constant declarations
        function definitions
        procedure definitions
        component declarations
 begin
        concurrent statement
        ...
        concurrent statement
end architecture-name;
```

COMPONENT DECLARATION

```
component component_name
      port ( signal_names : mode signal type;
             signal_names : mode signal type;
             …
             signal_names : mode signal type);
end component;
```

COMPONENT INSTANTIATION

```
label: component_name port map (signal1, signal2, …,signaln);
or,
label: component_name port map (port1 =>signal1, port2 =>signal2, …, portn =>signaln);
```

DATAFLOW TYPE STATEMENTS:

Simple concurrent assignment statement

```
signal_name <= expression;
```

Conditional concurrent assignment statement

```
signal_name <=
    expression when boolean-expression else
    expression when boolean-expression else
    …
   expression when boolean-expression else
    expression;
```

with-select statement

```
with expression select
     signal_name <= signal_value when choices,
                    signal_value when choices,
                     …
                     signal_value when choices;
```

Note that conditional concurrent assignment statement and with-select statement cannot be used in a process statement. Instead, in a process, one can use the the sequential conditional assignment statements if and case.

BEVAVIORAL TYPE STATEMENTS:

process statement
process(*signal_name, signal_name, …, signal_name*)
type_declarations
variable declarations
constant declarations
begin
sequential-statement
…
sequential-statement
end process;
Simple sequential assignment statement
signal_name <= expression;
Simple variable assignment statement
variable_name := expression;
if statement in its general form
if *boolean_ expression* then *sequential_statements*
elsif *boolean_ expression* then *sequential_statements*
…
elsif *boolean_ expression* then *sequential_statements*
else *sequential_statements*
end if;
Note that you may not use the else and/or the elsif.
case-when statement
case *expression* is
when *choices* => *sequential_statements*
…
when *choices* => *sequential_statements*
end case;
loop statement
loop
sequential_statement
…
sequential_statement
end loop;
for-loop statement
for *identifier* in *range* loop
sequential_statement
…
sequential_statement
end loop;
while statement
while *boolean_expression* loop
sequential_statement
…
sequential_statement
end loop;

Note that the if, case, loop, for, and while statements are called sequential statements and they can only be used in a process statement. Also note that each process is one concurrent statement.

If the "ieee.std_logic_arith.all" and "ieee.std_logic_unsigned.all" packages are included then + and – operators for addition and subtraction can be used for UNSIGNED binary, SIGNED binary, and STD_LOGIC_VECTOR types.

Concatenation operator & is used as follows: If A and B are 2 bit numbers then A&B is a four bit number with A being more significant.

BILKENT UNIVERSITY
Department of Electrical and Electronics Engineering
EEE102 Introduction to Digital Circuit Design
MidTerm Exam I
14-03-2005
Duration 110 minutes

Surname:_____

Name:_____

ID-Number:_____

Signature:_____

There are 7 questions of different weights. Solve all. Do not detach pages.

Q1	
Q2	
Q3	
Q4	
Q5	
Q6	
Q7	
TOTAL	

1. (16 points)
   a) Show that A·B = A·C does not imply that B = C.
   b) Show that NAND and NOR operators are not associative.
   c) Given that X·Y’ + X’·Y = Z, show that X·Z’ + X’·Z = Y.
   d) Convert the expression {[(A+B+A’·C’)·C+D]’+A·B’} into sum of minterms.

2. (12 points) Find all minimal sum expressions and all minimal product expressions for

$$F = \sum_{A,B,C,D}(0,2,3,4,8,10,11,12,14,15).$$

(Draw Karnaugh Maps, indicate prime implicants and distinguished 1s and 0s).

3. (16 points) Find all minimal sum expressions and all minimal product expressions for

$$F = \sum_{A,B,C,D}(0,2,3,4,12,13) + d(5,7,10,11,14).$$

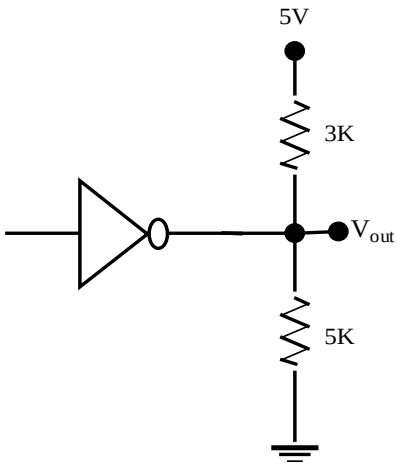
Determine which of these minimal expressions are equivalent. Explain why.
(Draw Karnaugh Maps, indicate prime implicants and distinguished 1s and 0s).

4. (12 points) Find all minimal sum expressions for

$$F = \prod_{A,B,C,D}(1,3,6,7,9,11,12,13,14,15).$$

Also find all minimal product expressions for F by using the methodology of finding minimal sums.
(Draw Karnaugh Maps, indicate prime implicants and distinguished 1s and 0s).

5. (16 points) A CMOS inverter working from 5V supply has a resistive load as shown below.



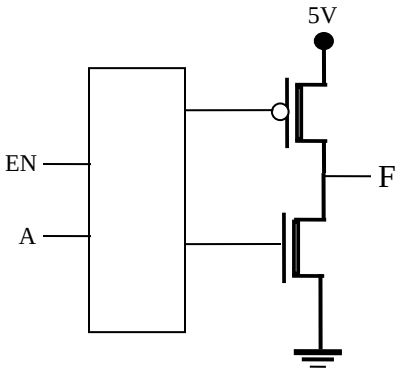
Suppose this CMOS inverter has the following specifications:

V_{IHmin} = 3.5 V, V_{ILmax} = 1.5 V , I_{ILmax} = -10 μA, I_{IHmax} = 15 μA.
The ON and OFF resistances of the NMOS and PMOS transistors are 1KΩ and 1000 KΩ respectively.

a) Find V_{out} for when it is HIGH and also for when it is LOW.
b) How many additional CMOS inverters of the same type can be connected to V_{out} ?

6. (12 points) Draw the internal circuit of a CMOS circuit (using NMOS and PMOS transistors) which has the logic function F = A'·B' + C'·D'.

7. (16 points) The rectangular block in the below drawing represents a logic circuit (logic block). Design this logic circuit so that if EN is 0 then F is in Hi-Z state and if EN is 1 then F = A'. In order words, you are to design the inside of the logic block so that the whole circuit becomes a three-state inverter. Make your design using gates (not transistors). Draw the whole circuit.



BILKENT UNIVERSITY
Department of Electrical and Electronics Engineering
EEE102 Introduction to Digital Circuit Design
MidTerm Exam I Solution
14-03-2005
Duration 110 minutes

1. (16 points)
   a) Show that A·B = A·C does not imply that B = C.
   b) Show that NAND and NOR operators are not associative.
   c) Given that X·Y' + X'·Y = Z, show that X·Z' + X'·Z = Y.
   d) Convert the expression {[(A+B+A'·C')·C+D]'+A·B'} into sum of minterms.

Solution:
1a)

	A	B	C	A·B	A·C
0	0	0	0	0	0
1	**0**	**0**	**1**	**0**	**0**
2	**0**	**1**	**0**	**0**	**0**
3	0	1	1	0	0
4	1	0	0	0	0
5	1	0	1	0	1
6	1	1	0	1	0
7	1	1	1	1	1

Note that in rows 0,1,2,3,4, and 7, A·B and A·C have the same values, but in rows 1 and 2, B and C are not equal.

1b) If nand operator is associative then [(A nand B) nand C] must be equal to [A nand (C nand B)].

(A nand B) nand C = [(A·B)'·C]' = A·B**+**C' (1)

A nand (C nand B) = [A·(B·C)']' = A'+B·C (2)

Expressions (1) and (2) are not equal because forexample if A = 0 and C = 1, then expression (1) is 0 but expression (2) is 1.

If nor operator is associative then [(A nor B) nor C] must be equal to [A nor (C nor B)].

(A nor B) nor C = [(A**+**B)'**+**C]' = (A**+**B)·C' (1)

A nor (C nor B) = [A**+**(B**+**C)']' = A'·(B+C) (2)

Expressions (1) and (2) are not equal because forexample if A = 0 and C = 1, then expression (1) is 0 but expression (2) is 1.

1c) XY'+X'Y=Z is given.

XZ'+X'Z = X(XY'+X'Y)' + X'(XY'+X'Y) substituting Z

= X(X'+Y)(X+Y') + X'XY' +X'X'Y

= X(X'X+X'Y'+YX+YY') + X'Y

= XY + X'Y

= (X+X')Y

= Y

XY'+X'Y=Z means $Z=X \oplus Y$

Shorter solution: $XZ'+X'Z = X \oplus Z = X \oplus (X \oplus Y) = (X \oplus X) \oplus Y$

$= 0 \oplus Y = Y$

1d) [(A+B+A'·C')·C+D]'+A·B'

= [AC+BC+D]'+AB'

= (A'+C')(B'+C')D' + AB'

= (A'B'+A'C'+B'C'+C')D'+AB'

= (A'B'+C')D'+AB'

= A'B'D'+ C'D'+AB'

= A'B'(C'+C)D'+(A'B'+A'B+AB'+AB)C'D'+AB'(C'D'+C'D+CD'+CD)

= A'B'C'D'+A'B'CD'+A'B'C'D'+A'BC'D'+AB'C'D'+ABC'D'
+AB'C'D'+AB'C'D+AB'CD'+AB'CD

= A'B'C'D'+A'B'CD'+A'BC'D'+AB'C'D'+ABC'D'
+AB'C'D+AB'CD'+AB'CD

Another method of solution: F = [(A+B+A'·C')·C+D]'+A·B' has the TT

A	B	C	D	F
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

We can then write the sum of minterms

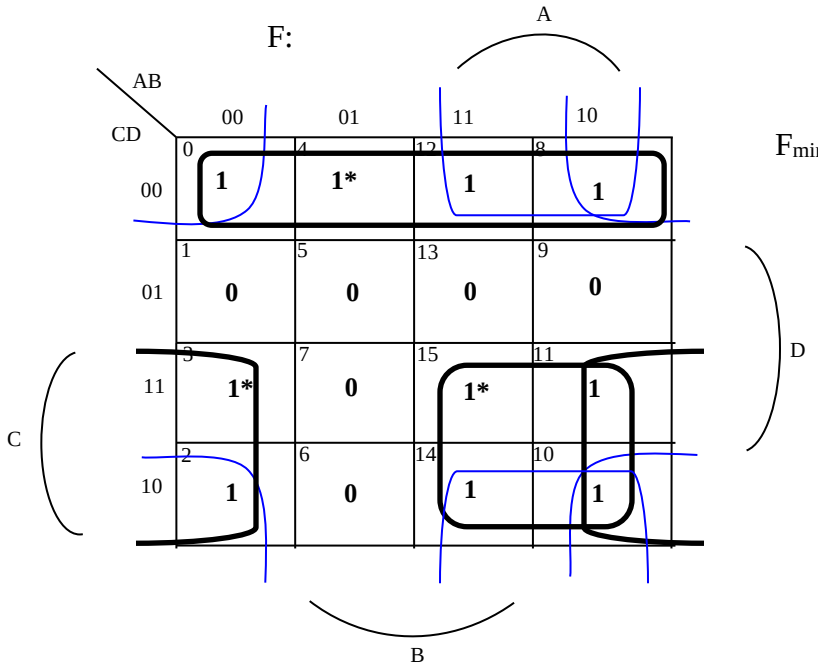
F = A'B'C'D'+A'B'CD'+A'BC'D'+AB'C'D'+AB'C'D+AB'CD'+AB'CD+ABC'D'

2. (12 points) Find all minimal sum expressions and all minimal product expressions for

$$F = \sum_{A,B,C,D}(0,2,3,4,8,10,11,12,14,15).$$

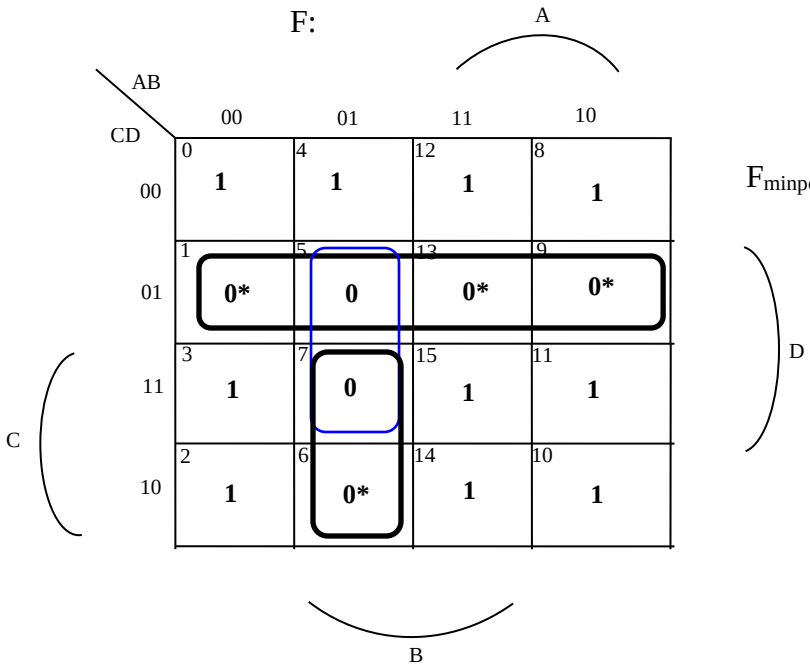
(Draw Karnaugh Maps, indicate prime implicants and distinguished 1s and 0s).

Solution:



F_{minsop} = C'D'+B'C+AC

F:



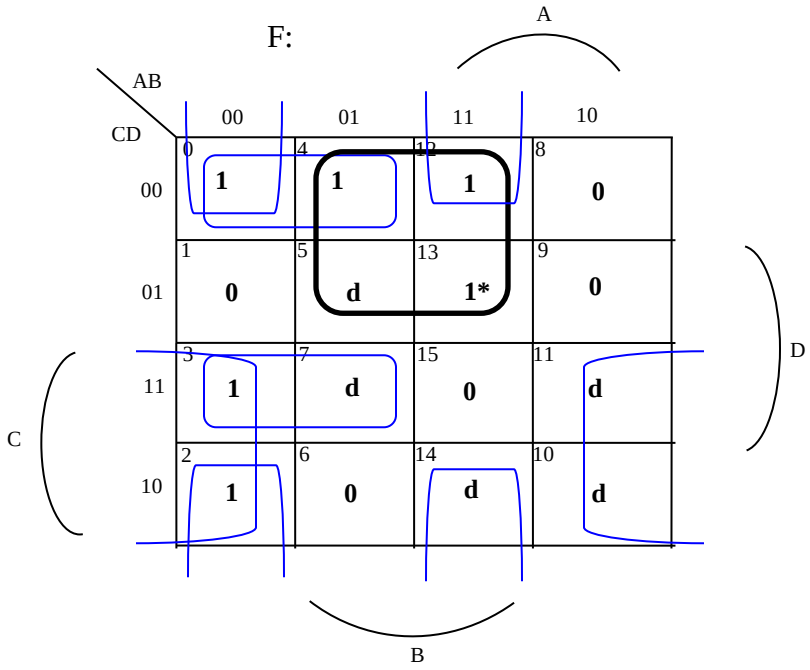
$F_{minpos} = (C+D')(A+B'+C')$

3. (16 points) Find all minimal sum expressions and all minimal product expressions for

$$F = \sum_{A,B,C,D}(0,2,3,4,12,13) + d(5,7,10,11,14).$$

Determine which of these minimal expressions are equivalent. Explain why. (Draw Karnaugh Maps, indicate prime implicants and distinguished 1s and 0s).

F:

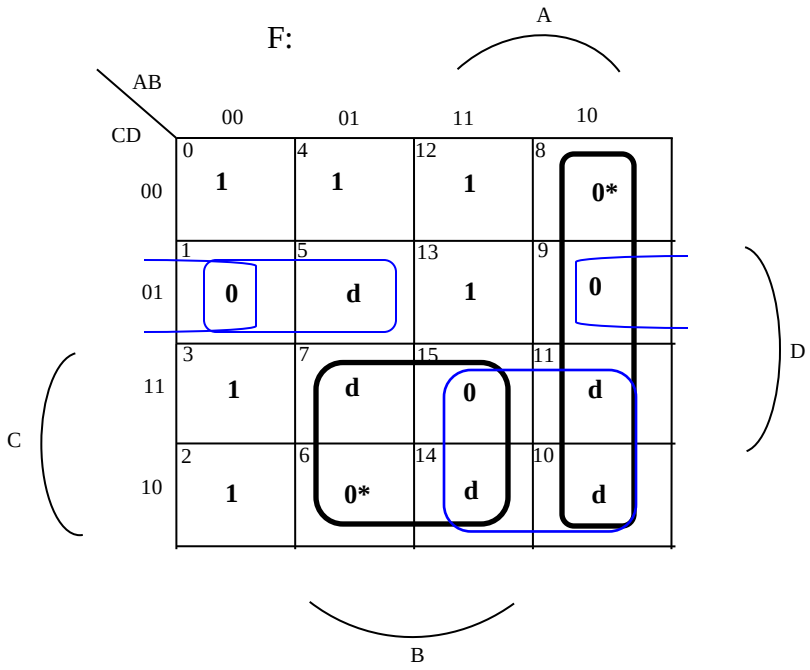


$F_{minsop1}$ = BC'+B'C+A'B'D'

5	7	10	11	14
1	0	1	1	0

$F_{minsop2}$ = BC'+B'C+A'C'D'

5	7	10	11	14
1	0	1	1	0



$F_{minpos1} = (B'+C')(A'+B)(A+C+D')$

5	7	10	11	14
0	0	0	0	0

$F_{minpos2} = (B'+C')(A'+B)(B+C+D')$

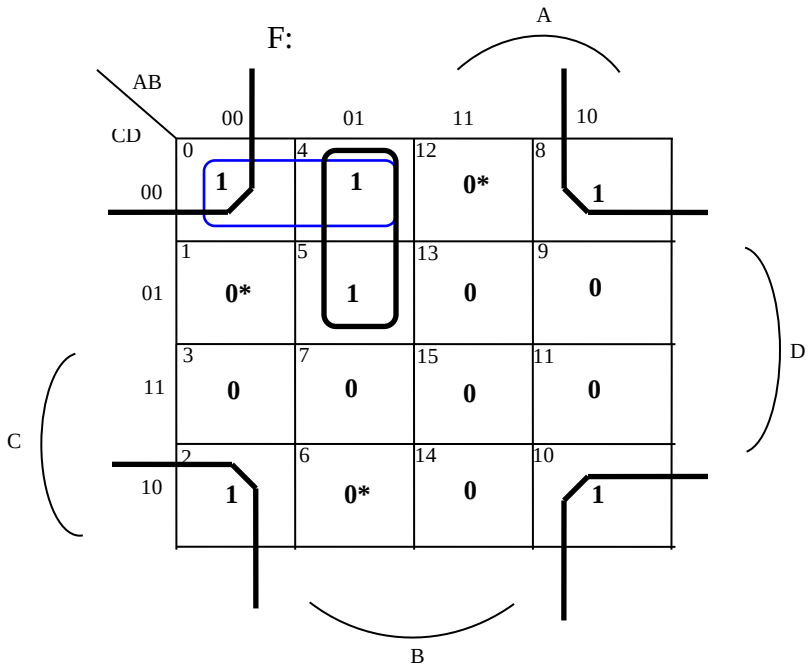
5	7	10	11	14
1	0	0	0	0

$F_{minsop1}$ and $F_{minsop2}$ are equivalent because don't cares have the same values . All the other solutions have different don't care values.

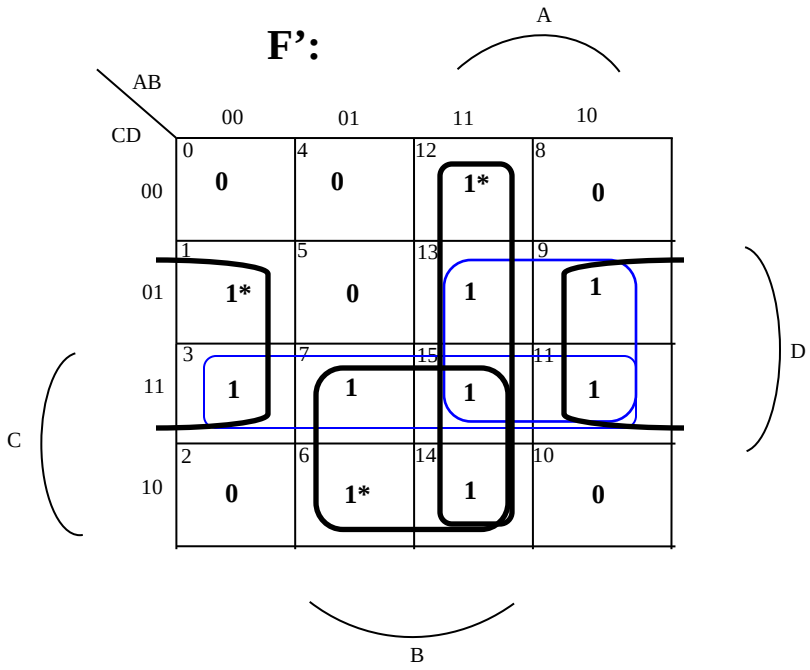
4. (12 points) Find all minimal sum expressions for

$$F = \prod_{A,B,C,D}(1,3,6,7,9,11,12,13,14,15).$$

Also find all minimal product expressions for F by using the methodology of finding minimal sums.
(Draw Karnaugh Maps, indicate prime implicants and distinguished 1s and 0s).



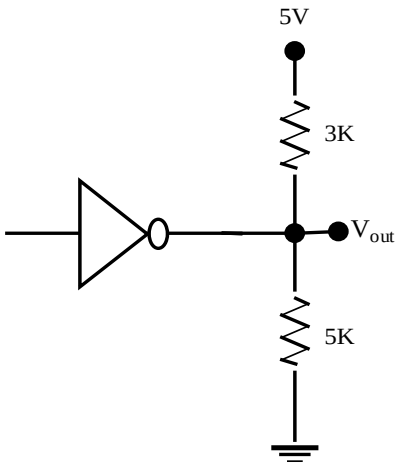
$F_{minsop} = B'D' + A'BC'$



$\mathbf{F'_{minsop}} = B'D + AB + BC$

$\mathbf{F_{minpos}} = (B+D')(A'+B')(B'+C')$

5. (16 points) A CMOS inverter working from 5V supply has a resistive load as shown below.



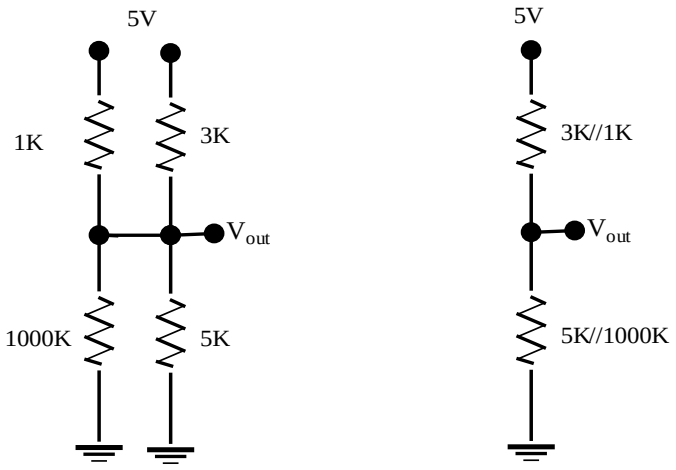
Suppose this CMOS inverter has the following specifications:

V_{IHmin} = 3.5 V, V_{ILmax} = 1.5 V , I_{ILmax} = -10 μA, I_{IHmax} = 15 μA.
The ON and OFF resistances of the NMOS and PMOS transistors are 1KΩ and 1000 KΩ respectively.

a) Find V_{out} for when it is HIGH and also for when it is LOW.
b) How many additional CMOS inverters of the same type can be connected to V_{out} ?

Solution:

a) Output HIGH:

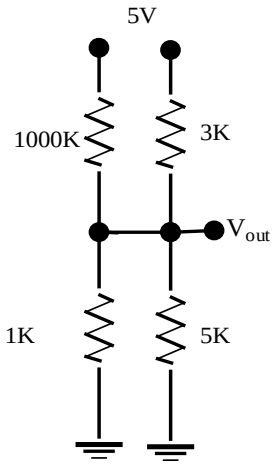


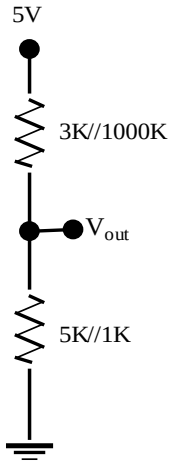
$3K // 1K = 0.75K$

$1000K // 5K \approx 5K$

$$V_{out} = 5V \times \frac{5}{5 + .75} = 4.35V$$

Output LOW:



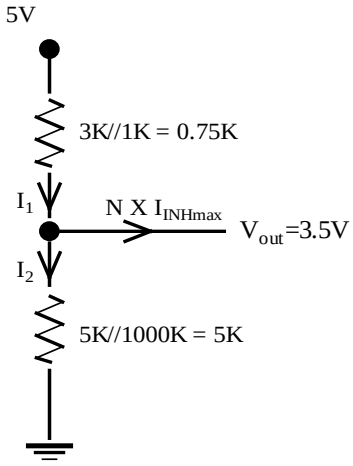


$3K // 1000K \approx 3K$

$1K // 5K \approx 0.833K$

$$V_{out} = 5V \times \frac{0.833}{3 + .833} = 1.09V$$

b) Output HIGH



$V_{out} = 3.5V$

$I_1 = \frac{5-3.5}{0.75K} = 2mA$

$I_2 = \frac{3.5}{5} = 0.7mA$

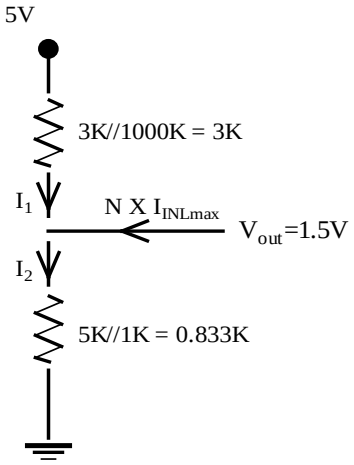
$N \times 15\mu A = I_1 - I_2 = 2 - 0.7 = 1.3mA$

$N = \frac{1.3mA}{15\mu A} = 86.7$

N must be an integer less than or equal to 86.7

Therefore take $N = 86$

Output LOW



$V_{out} = 1.5V$

$I_1 = \frac{5-1.5}{3K} = 1.167mA$

$I_2 = \frac{1.5}{0.833} = 1.8mA$

$N \times 10\mu A = I_2 - I_1 = 1.8 - 1.167 = 0.633mA$

$N = \frac{0.633mA}{10\mu A} = 63.3$

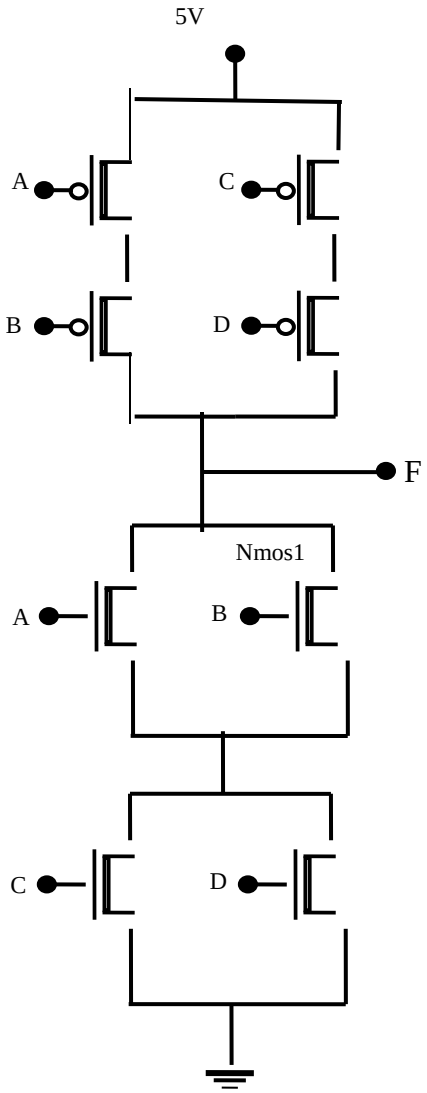
N must be an integer less than or equal to 63.3

Therefore take $N = 63$

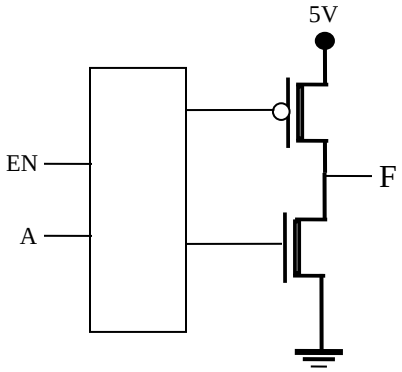
$Fanout = \min\{86, 63\} = 63$

6. (12 points) Draw the internal circuit of a CMOS circuit (using NMOS and PMOS transistors) which has the logic function F = A'·B' + C'·D'.

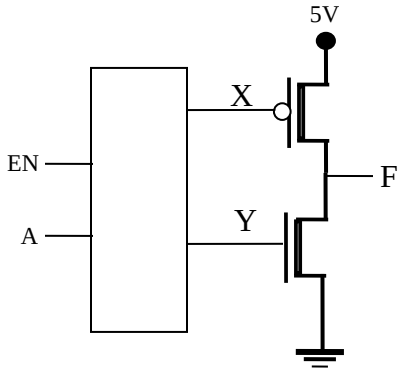
Solution: F = A'·B' + C'·D'= [(A+B).(C+D)]'



7. (16 points) The rectangular block in the below drawing represents a logic circuit (logic block). Design this logic circuit so that if EN is 0 then F is in Hi-Z state and if EN is 1 then F = A'. In order words, you are to design the inside of the logic block so that the whole circuit becomes a three-state inverter. Make your design using gates (not transistors). Draw the whole circuit.



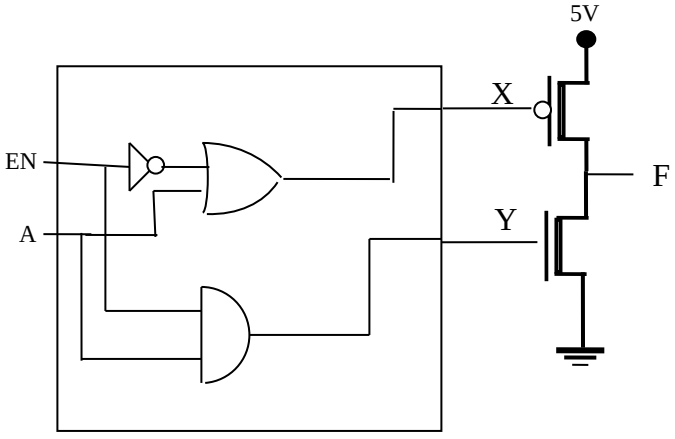
Solution:



EN	A	X	Y
0	0	1	0
0	1	1	0
1	0	0	0
1	1	1	1

X=EN'+A

Y=EN·A



BILKENT UNIVERSITY
Department of Electrical and Electronics Engineering
EEE102 Introduction to Digital Circuit Design
Midterm Exam-1

15-11-2006

Duration 120 minutes

Surname: _____

Name: _____

ID-Number: _____

Signature: _____

There are 6 questions. +Solve all.
Do not detach pages. Show all your work.

Q1	
Q2	
Q3	
Q4	
Q5	
Q6	
Total	

Q1. (20 points)

a) (8 pts.) Draw using simple gates, the internal circuitry of a 2-to-4 binary decoder with active-low enable.

b) (6pts.) Draw how a square wave is displayed on the oscilloscope screen if the probe is properly compensated, under-compensated or over-compensated.

c) (6 pts.) For the truth table given below draw a circuit which implements it. F is the output and A, B, C are the inputs.

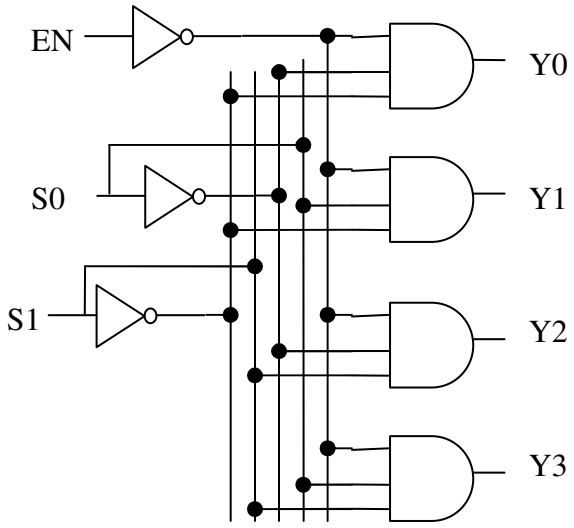
A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

SOLUTION:

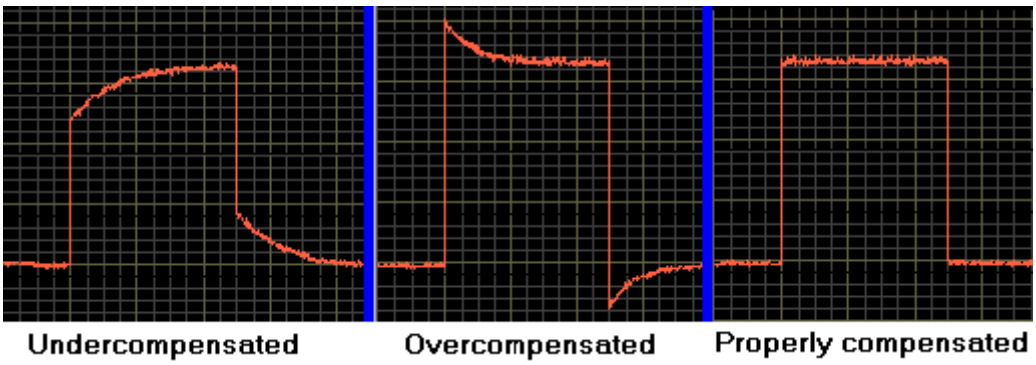
a)

The functions expressing the input output relations in a 2-to-4 binary decoder with enable are

Y0 = EN' and S1' and S0'
Y1 = EN' and S1' and S0
Y2 = EN' and S1 and S0'
Y3 = EN' and S1 and S0

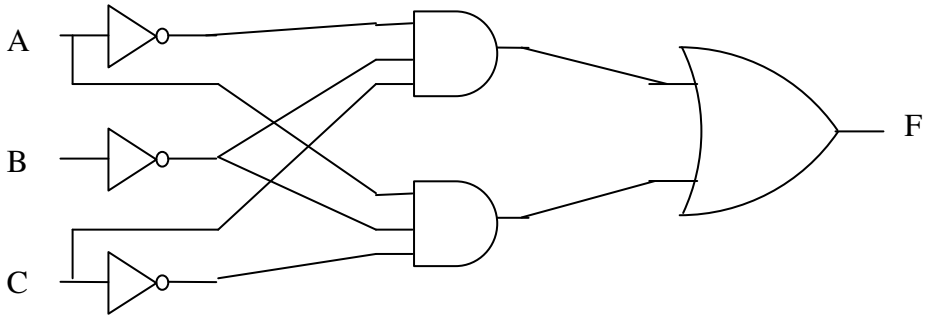


b)



c)

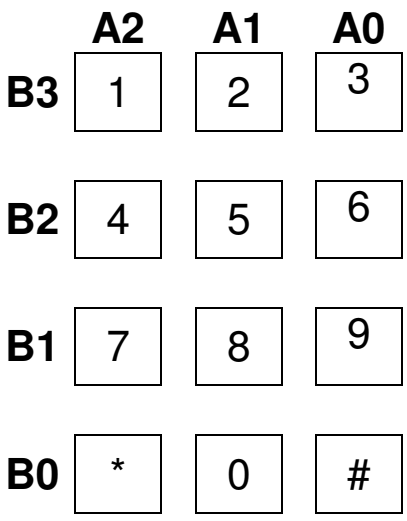
F = A'B'C + AB'C'

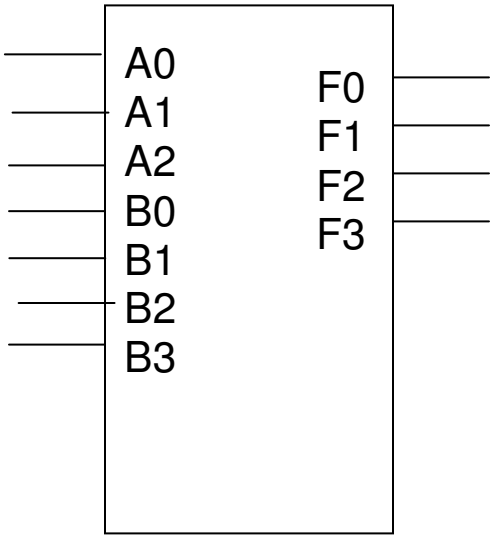


Q2. (20 points)

a) (10 pts.) Write the truth table of an encoder which encodes the pressed key in the following keypad. Assume that more than one key is not pressed at the same time. The keypad works as follows: For example, when the key 1 is pressed, A2 and B3 are asserted, or when key 5 is pressed, A1 and B2 are asserted.
This 7-to-4 encoder outputs the binary equivalent of the pressed number. When * key is pressed, the encoder outputs the binary equivalent of 10_{10} and when # key is pressed, the encoder outputs the binary equivalent of 11_{10}. If no key is pressed, all outputs are "1". The pin diagram of the 7-to-4 encoder that you will design is shown below.

b) (10 pts.) Describe the above encoder using VHDL





SOLUTION:

a)

A2	A1	A0	B3	B2	B1	B0	F3	F2	F1	F0
0	0	1	0	0	0	1	1	0	1	1
0	0	1	0	0	1	0	1	0	0	1
0	0	1	0	1	0	0	0	1	1	0
0	0	1	1	0	0	0	0	0	1	1
0	1	0	0	0	0	1	0	0	0	0
0	1	0	0	0	1	0	1	0	0	0
0	1	0	0	1	0	0	0	1	0	1
0	1	0	1	0	0	0	0	0	1	0
1	0	0	0	0	0	1	1	0	1	0
1	0	0	0	0	1	0	0	1	1	1
1	0	0	0	1	0	0	0	1	0	0
1	0	0	1	0	0	0	0	0	0	1
All others							d	d	d	d

b)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity keypadencoder is
port(A:in std_logic_vector (2 downto 0);
```

```
        B:in std_logic_vector (3 downto 0);
        F:out std_logic_vector (3 downto 0));
end keypadencoder;

architecture Behavioral of keypadencoder is

begin
process(A,B)
begin
  if A="000" and B="0000" then F<="1111";

        elsif A="001" and B="0001" then F<="1011";
        elsif A="001" and B="0010" then F<="1001";
        elsif A="001" and B="0100" then F<="0110";
        elsif A="001" and B="1000" then F<="0011";

        elsif A="010" and B="0001" then F<="0000";
        elsif A="010" and B="0010" then F<="1000";
        elsif A="010" and B="0100" then F<="0101";
        elsif A="010" and B="1000" then F<="0010";

        elsif A="100" and B="0001" then F<="1010";
        elsif A="100" and B="0010" then F<="0111";
        elsif A="100" and B="0100" then F<="0100";
        elsif A="100" and B="1000" then F<="0001";

        else F<="1111";
  end if;

end process;

end Behavioral;
```

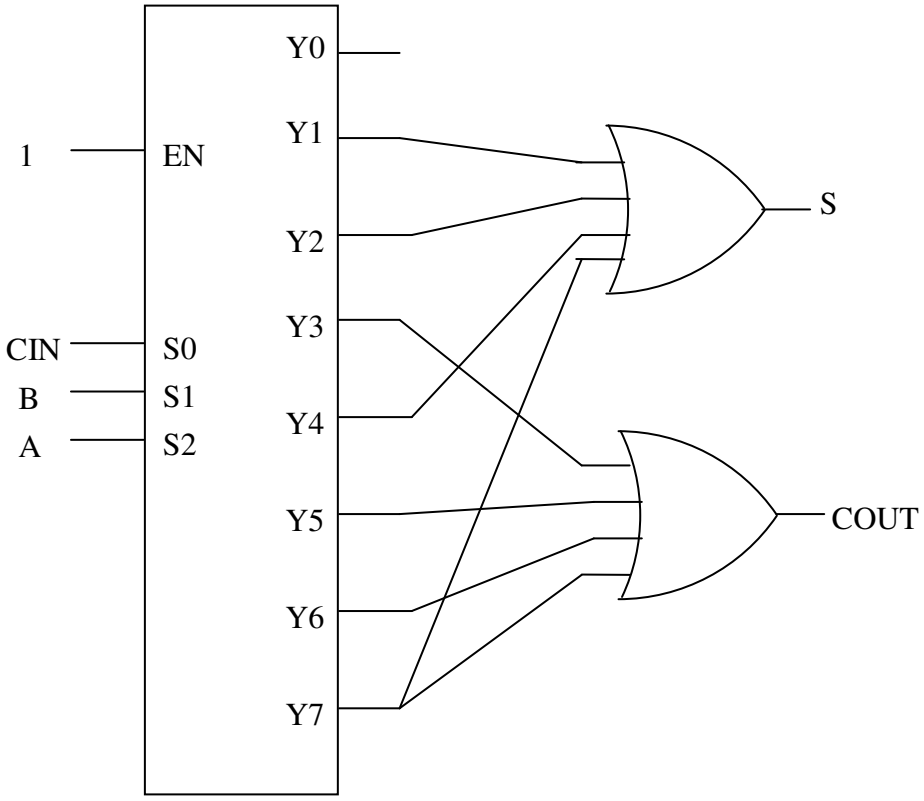
For the else F<="1111"; statement you can use any other value because these input combinations do not occur. In order to make the design process simpler and to achieve a smaller design you can also write else F<="----"; meaning don't care. The sign - is the same as the d that we use in class.

Q3. (15 points)
Implement a single bit full adder using a generic 3-to-8 decoder and additional simple gates as necessary.

SOLUTION:

A	B	CIN	S	COUT
0	0	0	0	0
0	0	1	1	0

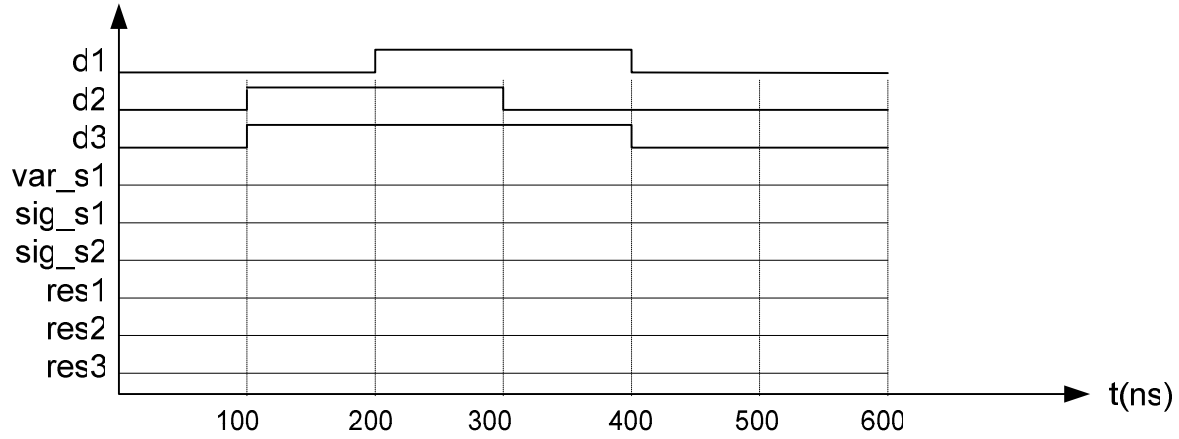
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



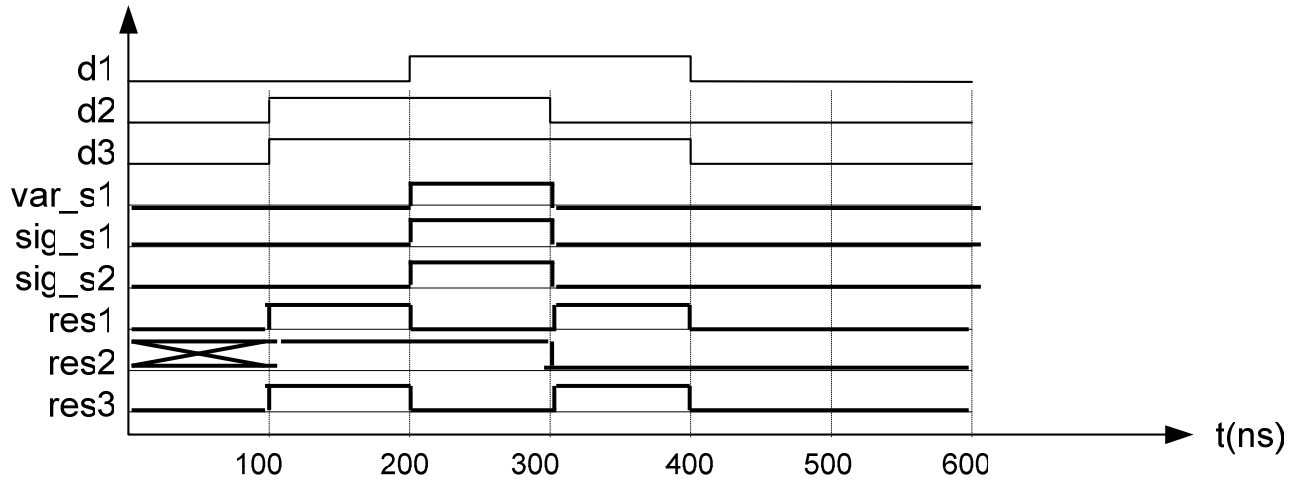
Q4. (15 points)
Draw the time waveforms of var_s1, sig_s1, sig_s2, res1, res2, and res3 for t≥0 for the given time waveforms of d1, d2, and d3 for a circuit described by the following VHDL code.

```
library ieee;
use ieee.std_logic_1164.all;

entity sig_var is
port(d1, d2, d3:  in std_logic;
      res1, res2, res3: out std_logic);
```

```
end sig_var;

architecture behv of sig_var is

  signal sig_s1: std_logic;
  signal sig_s2: std_logic;

begin

  proc1: process(d1,d2,d3)
   variable var_s1: std_logic;
  begin
      var_s1 := d1 and d2;
      res1 <= var_s1 xor d3;
  end process;

  proc2: process(d1,d2,d3)
  begin
      sig_s1 <= d1 and d2;
      res2 <= sig_s1 xor d3;
  end process;

  proc3: process(d1,d2,d3,sig_s2)
  begin
      sig_s2 <= d1 and d2;
      res3 <= sig_s2 xor d3;
  end process;

 end behv;
```



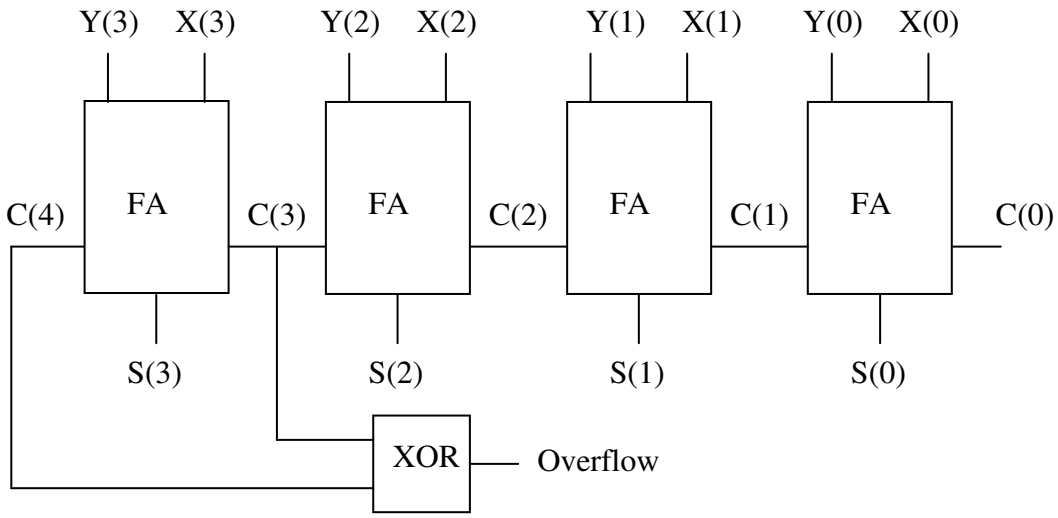
SOLUTION:



Q5. (15 points)
Write a VHDL code to obtain a 4-bit adder with overflow detector using the VHDL module FA. FA module is a Full Adder and it is already in the library with the following entity declaration.

```
entity FA is
      port (A, B, CIN : in std_logic;
            S, COUT : out std_logic);
end FA;
```

SOLUTION: We have not specified in the problem whether we want a design for two's complement 4-bit numbers. Below is the design for two's complement 4-bit numbers.



The same circuit can be used for unsigned 4-bit binary numbers except that in that case Overflow = C(4).
A different design can also be made for sign-magnitude 4-bit numbers in which case the design is more complicated.

Code for FA which you did not have to write in the exam:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity FA is
   Port ( A,B,CIN : in std_logic;
          S,COUT : out std_logic);
end FA;

architecture Behavioral of FA is
begin
S <= A xor B xor CIN;
COUT <= (A and B) or (CIN and (A or B));
end Behavioral;
```

Code for the 4-bit adder for two's complement numbers:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
-- four bit adder for two's complement numbers

entity foutbitadderwo is
   Port ( X : in std_logic_vector (3 downto 0);
          Y : in std_logic_vector (3 downto 0);
          S : out std_logic_vector (3 downto 0);
          O : out std_logic);
end foutbitadderwo;

architecture Behavioral of foutbitadderwo is
component FA
   Port ( A : in std_logic;
          B : in std_logic;
          CIN : in std_logic;
          S : out std_logic;
          COUT : out std_logic);
end component;
signal C: std_logic_vector (4 downto 0) :="00000";
begin
L1: FA port map(X(0),Y(0),C(0),S(0),C(1));
L2: FA port map(X(1),Y(1),C(1),S(1),C(2));
L3: FA port map(X(2),Y(2),C(2),S(2),C(3));
L4: FA port map(X(3),Y(3),C(3),S(3),C(4));
O<=C(3) xor C(4);
end Behavioral;
```

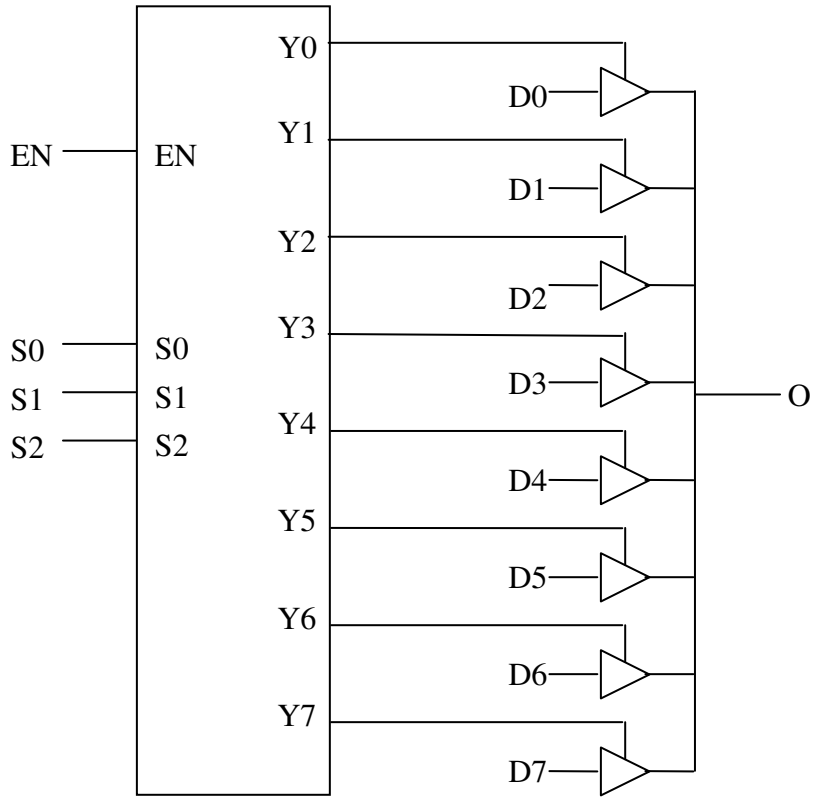
It is also possible to instantiate the FAs using a for-generate loop which is given below for your information.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
-- four bit adder for two's complement numbers

entity adderpro is
   Port ( X : in std_logic_vector (3 downto 0);
          Y : in std_logic_vector (3 downto 0);
          S : out std_logic_vector (3 downto 0);
          O : out std_logic);
end adderpro;

architecture Behavioral of adderpro is
component FA
   Port ( A : in std_logic;
          B : in std_logic;
```

```
        CIN : in std_logic;
        S : out std_logic;
        COUT : out std_logic);
end component;
signal C: std_logic_vector (4 downto 0) :="00000";
begin
        G1:for i in 0 to 3 generate
        begin
        U1: FA port map(X(i),Y(i),C(i),S(i),C(i+1));
        end generate;
        O<=C(3) xor C(4);
end Behavioral;
```

Q6. (15 points)
Design and draw a 8-to-1 multiplexer with enable, using a 3-to-8 decoder with enable, and 8 three-state buffers. When the multiplexer that you design is disabled, its output must be at high Z. Label all pins, inputs, outputs etc. properly.

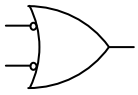
SOLUTION:



S0, S1, S2 are the selects of the 8-to-1 mux, EN is its enable, O is its output and D0,…,D7 are its data inputs. When EN is '0' the output is at High-Z.

MIDTERM 1 SOLUTIONS

Q1. (Total of 33 points)
a) (4p) Show using DeMorgan's Theorem that

 **is a symbol for a NAND gate.**

De Morgan's Theorem states that

$(A \cdot B)' = A' + B'$.

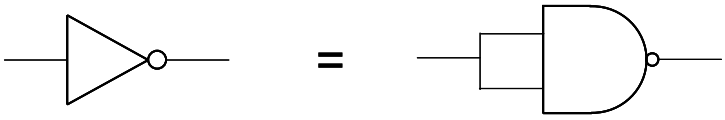
The left-hand side of this equation is the standard NAND definition, and the right-hand side is the alternative representation depicted in the above symbol.

b) (6p) Prove using the canonical SOP representation of a combinational logic function that NAND gates are sufficient to implement any combinational logic circuit.

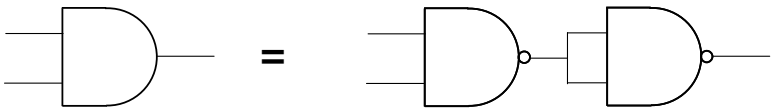
Method I:

Any combinational logic function can be represented in Canonical SOP form. It is trivially straightforward to implement the Canonical SOP form using inverters (to implement complemented literals), AND gates (to implement the products), and OR gates (to implement sums). It is sufficient to show that we can implement each of inverter, AND, and OR gate using NAND gates.

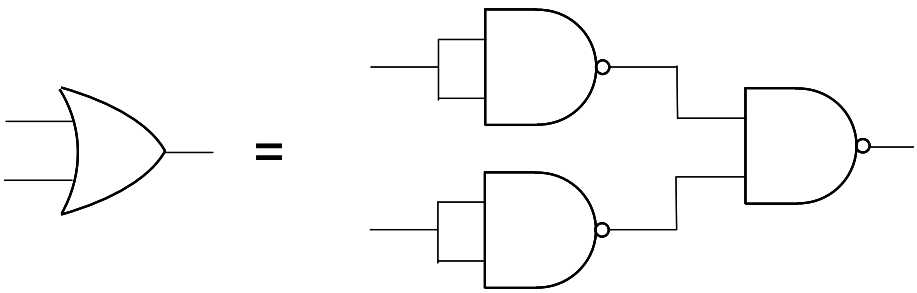
Inverter:



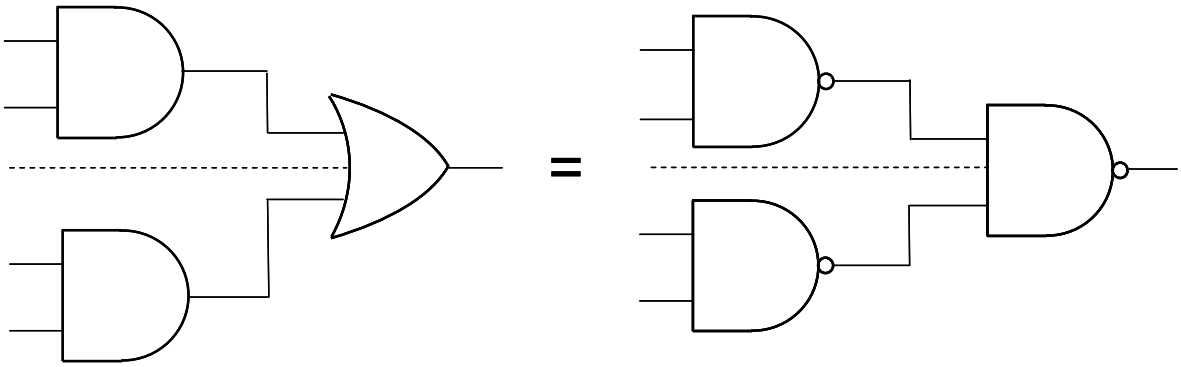
AND:



OR:



Method II: Show inverter implementation as in Method II. Instead of showing AND and OR implementation independently, just show the implementation of AND-OR combination used in Canonical SOP form.



c) (4p) Find the complement of the function F = A + BC'(D + E+ 1). Do not simplify the expression.

We apply De Morgan's Rule repeatedly.

$F' = (A + BC'(D+E+1))' = A' \cdot (BC'(D+E+1))' = A' \cdot (B' + C + (D+E+1)')$
$= A' \cdot (B' + C + (D' \cdot E' \cdot 0))$

d) (4p) Find the dual of the function F = A + BC'(D + E+ 1). Do not simplify the expression.

$F^D = (A + (BC'(D+E+1)))^D = (A \cdot (B + C' + (D \cdot E \cdot 0)))$

e) (4p) Expand the function F = A B + CD with respect to A using Shannon's theorem.

$F(A,B,C) = A' \cdot F(0,B,C) + A \cdot F(1,B,C) = A' \cdot (C \cdot D) + A \cdot (B + C \cdot D)$

f) (5p) Add out the function F = A + BC + D to obtain a POS form.

$F = A + BC + D = (A + B)(A+C) + D = (A+B+D)(A+C+D)$

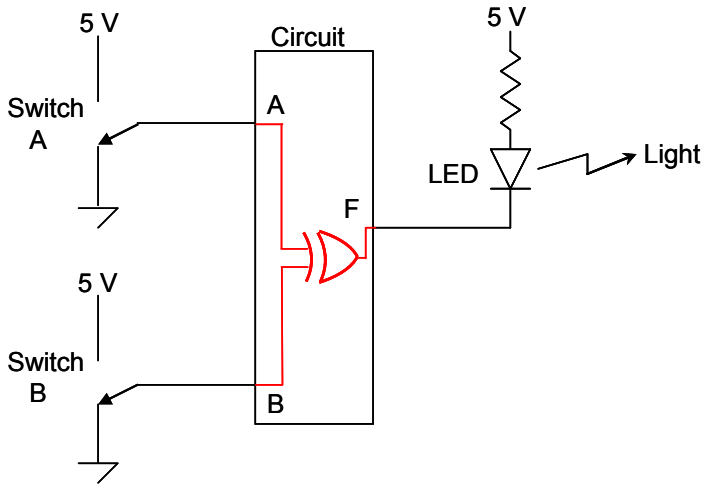
g) (6p) The Boolean variables A, B, and C have values that we don't know, but we know that AB = AC. If C = 0, what can we say about B.

If A=1, then B=C, which means B=0.

If A=0, then both sides of AB = AC will be 0, regardless of the values of B and C. In this case, we cannot say anything about the value of B. (It can be 0 or 1)

In short, without knowing the value of A, we cannot say anything about the value of B.

Q2. (15 points)
A "two-way switch" system is used at homes to turn a light ON or OFF from two switches located in two different places.



If switch A is pulled UP then the signal A becomes 1 and if it is pulled DOWN then the signal A bocomes 0. Similarly for switch B.

Now suppose switch A is located at the bottom of a stairway and switch B is located at the top. The LED (Light Emitting Diode) is used to illuminate the stairway. Assume, as a scenario, that the the LED is OFF at the beginning. A person who wants to climb up the stairs toggles switch A to turn the light ON. He then climbs up the stairs and at the top he toggles switch B to turn the light back OFF. This scenario is given as an example to explain how the system is used.

Design and draw the inside of the "Circuit" using as few gates as possible.

Notes: 1) For the LED to be ON, F must be 0. To turn it OFF, F must be 1.
2) "Toggle" means "change the position of the switch".
3) The initial positions of the switches are unknown although they are drawn in the above figure as at DOWN position.

Assume that both switches are DOWN, and LED is OFF at the beginning. This gives "*F = 1 when A=0 and B=0*". Next, we should switch on the light by toggling either one of A or B. This gives "*F=0 when A=1 and B=0*", and "*F=0 when A=0 and B=1*". Finally, we should switch off when the light is ON by toggling either one of A or B. This gives "*F=1 when A=1 and B=1*" and "*F=1 when A=0 and B=0*". The latter was already covered by the initial case. Combining the givens into a truth table, we have:

A	B	F
0	0	1
0	1	0
1	0	0
1	1	1

This is the truth table of an XNOR (equivalence) gate.

With the same reasoning and different initial conditions (e.g. "*both switches DOWN and LED is ON*", or "*one of the switches UP, one of the switches DOWN, and LED is ON*"), we will end up with the truth table of an XOR gate.

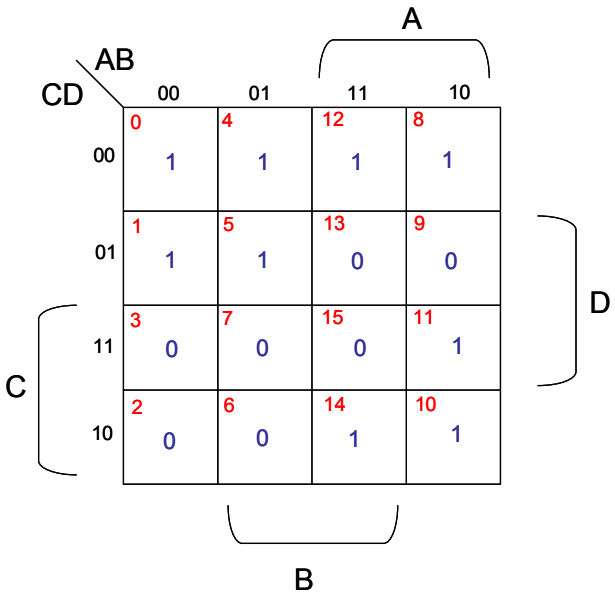
Note that in practice initial conditions do not matter: We build the circuit by using one XOR gate or one XNOR gate, and if the LED is ON, we switch it off by toggling one of the switches.

Q3. (Total of 24 points)

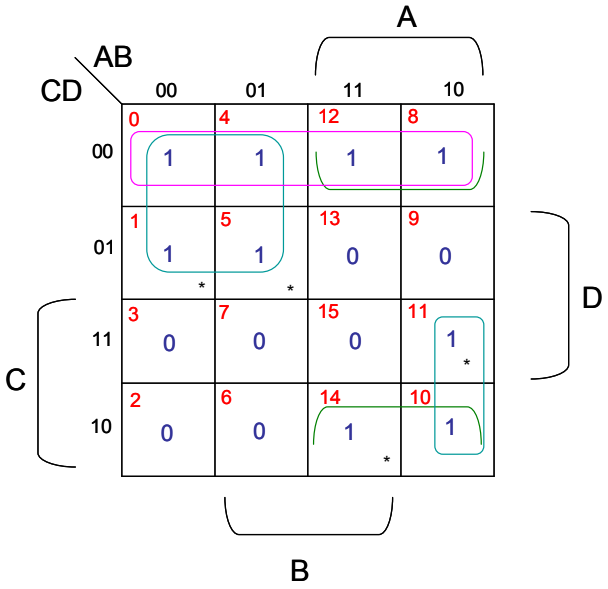
Use the Karnaugh Map method to answer the following questions for the function

F = A'B'C'D' + A'BC'D' + ABC'D' + AB'C'D' + A'B'C'D + A'BC'D + AB'CD + ABCD' + AB'CD'

a) (5p) Draw the Karnaugh map for F



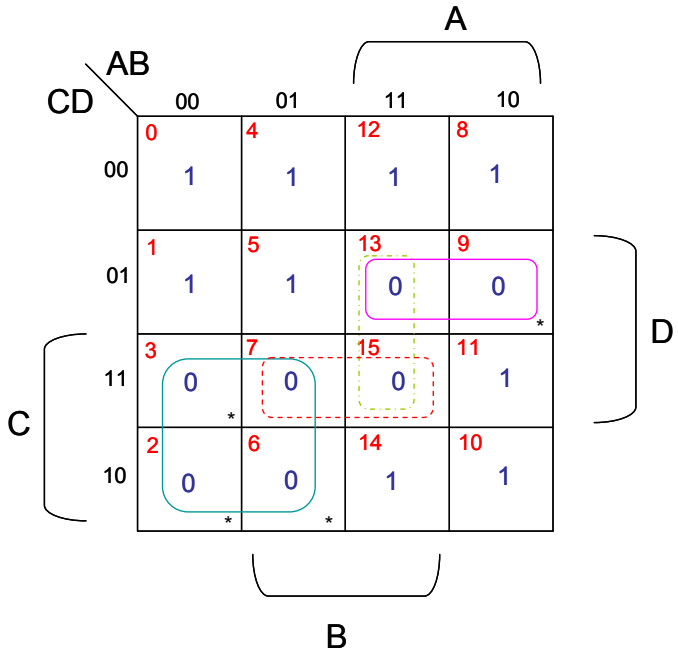
b) (7p) Find a minimal SOP expressions for F.



In this case, essential prime implicants cover all ones.

F = A′ C′ + A D′ + A B′ C

c) (7p) Find a minimal POS expressions for F.



In this case, after essential prime implicants are chosen, only one 0-cell is left uncovered (cell 15). This cell can be covered by the maxterm (A′ + B′ + D′) which is depicted with dash-dot in the graph, or by the maxterm (B′ + C′ + D′) which is depicted with dash in the graph. Both choices are equally good.

F = (A + C′) (A′ + C + D′) (A′ + B′ + D′)

or

F = (A + C′) (A′ + C + D′) (B′ + C′ + D′)

d) (5p) Which of the above found expressions would you use for implementation. Why?

SOP form has 3 terms and 7 literals. POS form has 3 terms and 8 literals. We would use SOP form, since it has no more terms and fewer literals, thus would result in fewer or smaller gates.

Note: If your Karnaugh map is not correct, you may not get points for the rest of your solution.

Q4. (20 points)
An odd-parity module (entity) "odd_parity_8bit" for 8-bit binary numbers is already written using VHDL and saved.
Write VHDL code for the entity named "odd_parity_16bit" which describes an odd_parity circuit for 16-bit binary numbers. Use "odd_parity_8bit" module in your code.

The entity declaration of "odd_parity_8bit" is as follows:

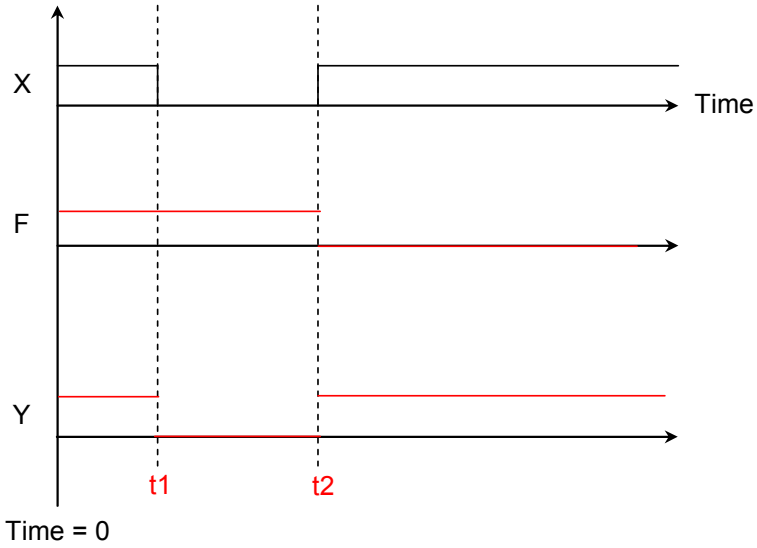
```
entity odd_parity_8bit is
        port (A:in std_logic_vector (7 downto 0);
              F: out std_logic);
end odd_parity_8bit;
```

Notes: 1) What "odd_parity_8bit" does: If the number of 1 bits in A is odd then F = 1, else F = 0.
2) If X is declared as std_logic_vector (7 downto 0) then X(5 downto 1) is a 5-bit number picking the appropriate subrange of X.
3) It is not important for this exam's purposes to include all necessary libraries.
4) Do not write "odd_parity_16bit" from scratch but make use of the previously written and saved "odd_parity_8bit".

```
entity odd_parity_16bit is
     port (A:in std_logic_vector (15 downto 0);
           F: out std_logic);
end odd_parity_16bit;

architecture structural of odd_parity_16bit is

component odd_parity_8bit
     port (A:in std_logic_vector (15 downto 0);
           F: out std_logic);
end component;

signal F8H, F8L: std_logic;

begin
  label1: odd_parity_8bit port map(A(15 downto 8), F8H);
  label2: odd_parity_8bit port map(A(7 downto 0), F8L);

  F <= F8H xor F8L;
End structural;
```

Q5. (15 points)
Draw the timing diagram for the circuit described by the following VHDL code for the input signal waveform given below.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity simple is
       port(X:in std_logic;
                F:out std_logic);
end simple;

architecture Behavioral of simple is
signal Y:std_logic :='1';

begin
process(X)
begin
       Y<=X;
       F<=Y;
end process;

end Behavioral;
```



Note that Y is not in the sensitivity list of the process. The process will be executed once at t=0, and then whenever the value of X changes, namely at t=1 and t=t2. Since X is not in the left hand side within the process, the process will be executed at most once for each t.

Remember that the new values of the assigned signals become effective after the process call terminates. For this reason, the statement F <= Y will store the ***old value*** of Y to F each time it executes. At t=0, the old value of Y is 1, due to the initialization statement. At t=t1, the old value of Y is 1. At t=t2, the old value of Y is 0.

Q6. (Total of 18 points)
Implement the function F = AB + B'C

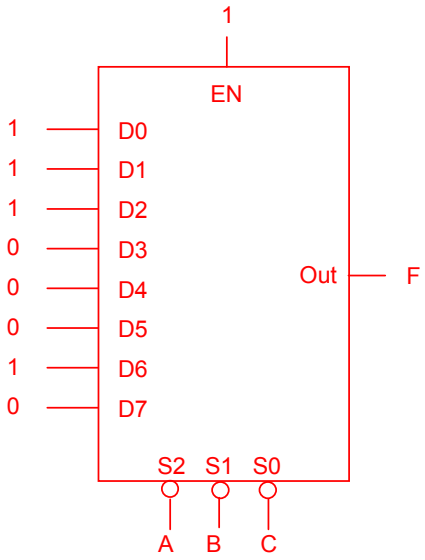
A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

a) (6p) using one 8-to-1 multiplexer with active_low selects (control inputs) only, (You are not allowed to use any extra gate. Draw your solution and label all inputs and outputs.)

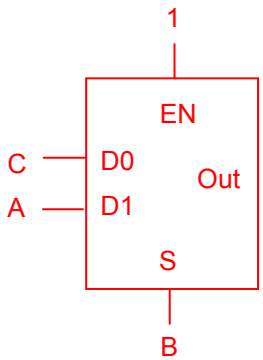
We need to reorder the inputs in order to compensate for the active-low select inputs. The reordering is:

New_index = 7 - TT row index

This is the same thing as applying the truth table "upside-down".



b) (6p) using one generic 2-to-1 multiplexer only, and
(You are not allowed to use any extra gate. Draw your solution and label all inputs and outputs.)



c) (6p) using one 74x138 3-to-8 decoder and a single multi-input gate.