18-12-2008

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

Surname: _____

Name: _____

ID-Number: _____

Signature: _____

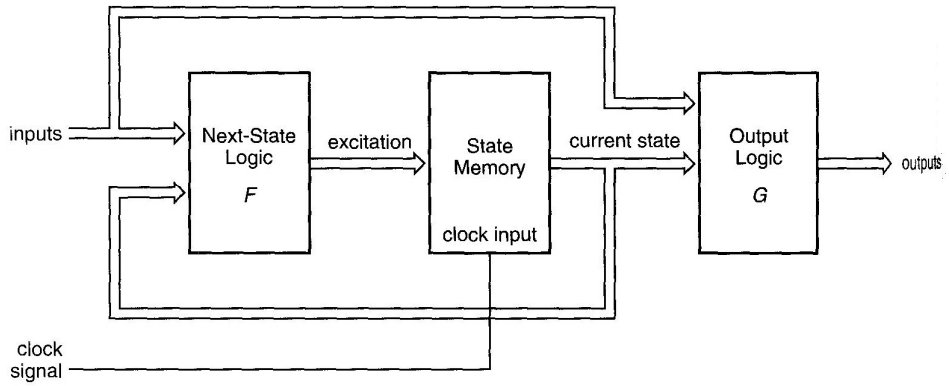
Duration is 120 minutes. Solve all 5 questions. Show all your work.

Q1 (12 points)	
Q2 (22 points)	
Q3 (22 points)	
Q4 (22 points)	
Q5 (22 points)	
Total	

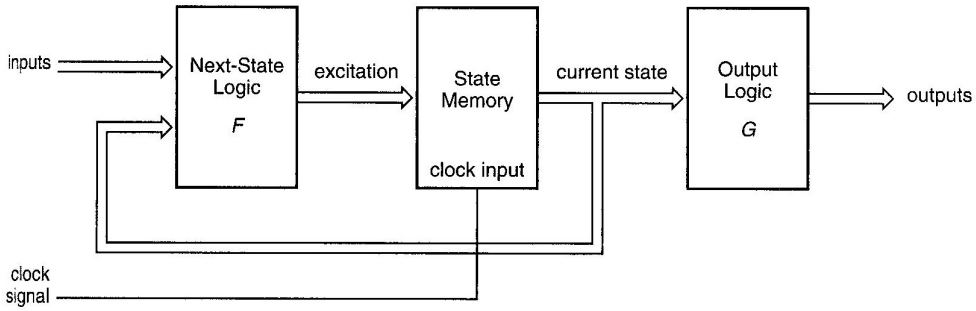
Question 1-(12 pts):

(a) (4 pts) Draw the block diagram of a synchronous finite-state machine, and explain the difference between Mealy and Moore machines. Make sure to label all of the blocks properly.

Clocked synchronous state-machine structure (Mealy machine)



Clocked synchronous state-machine structure (Moore machine)

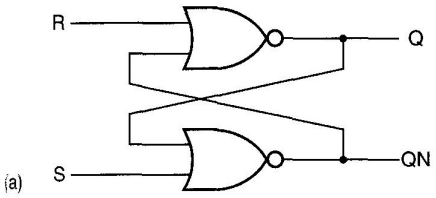


The outputs for Mealy machines depend on the current state as well as on the input directly. For Moore mchines, the outputs depend on the current state only.

(b) (4 pts) What is the fundamental difference between a flipflop and a latch?

Flipflops are edge-triggered while latches are level-sensitive.

(c) (4 pts) Draw the gate-level circuit diagram of an S-R latch and explain how it works.



S-R latch circuit design

(b)

S	R	Q	QN
0	0	last Q	last QN
0	1	0	1
1	0	1	0
1	1	0	0

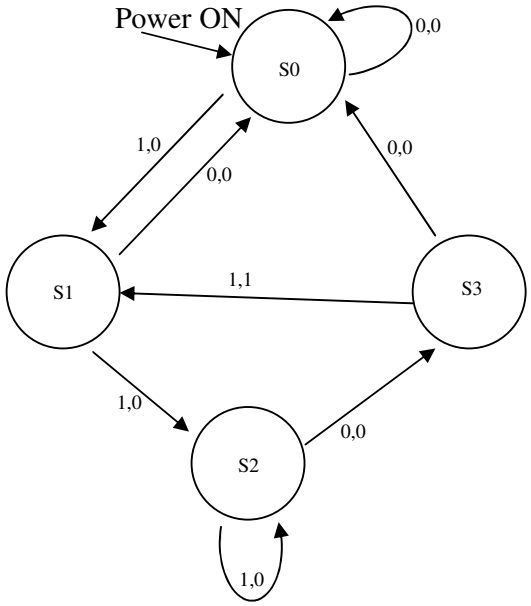
S-R latch function table

Question 2-(22 pts):

You are to design an FSM that recognizes the occurence of a particular sequence of bits. This "sequence-recognizer" has one input X and one output Z. The circuit is to recognize the occurence of the sequence of bits 1101 on X by making Z equal to 1 when the previous three inputs to the circuit were 110 and the current input is a 1. Otherwise, Z equals 0. Design and draw this FSM using D flip flops with active low Preset and Clear controls. Show all steps of FSM design procedure. Note for clarification that the LSB in the sequence 1101 is the present value of X.

Solution:

States:
S0: Power ON state (Also not enough information, or start of seeking a new sequence)
S1: First 1 of the sequence is received.
S2: Two 1s in a row are received.
S3: 110 is received.



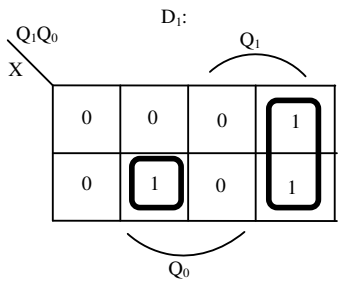
We need 2 flip flops.

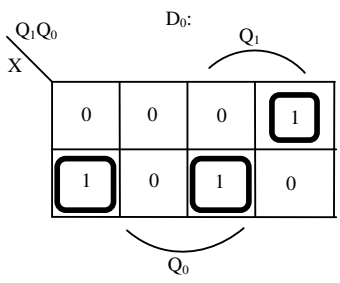
State encoding:

	Q1	Q0
S0	**0**	**0**
S1	**0**	**1**
S2	**1**	**0**
S3	**1**	**1**

Next state table:

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



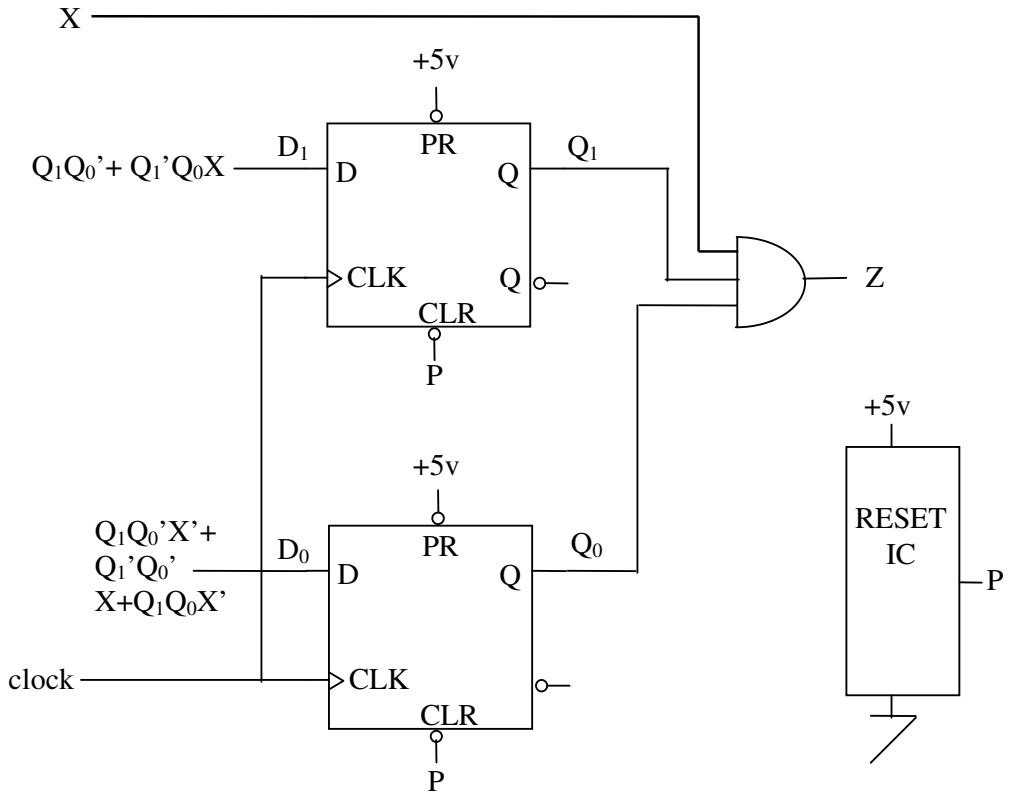


$D_1 = Q_1Q_0' + Q_1'Q_0X$ $D_0 = Q_1Q_0'X' + Q_1'Q_0' X + Q_1Q_0X$

Output table:

Q1	Q0	X	Z
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

$Z=Q_1Q_0X$



Another solution:

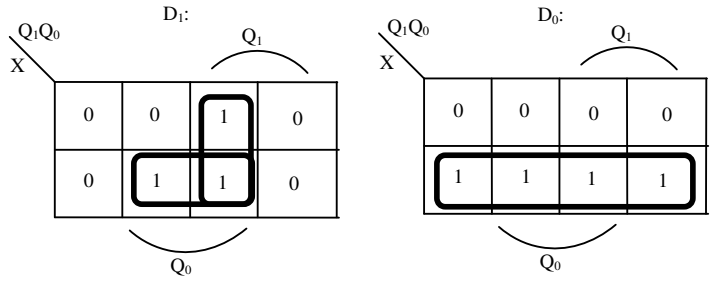
State encoding:

	Q1	Q0
S0	0	0
S1	0	1
S2	1	1
S3	1	0

Next state table:

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

0	**0**	**1**	**0**	**1**
0	**1**	**0**	**0**	**0**
0	**1**	**1**	**1**	**1**
1	**0**	**0**	**0**	**0**
1	**0**	**1**	**0**	**1**
1	**1**	**0**	**1**	**0**
1	**1**	**1**	**1**	**1**



$D_1 = Q_1Q_0 + Q_0X$ $D_0 = X$

Output table:

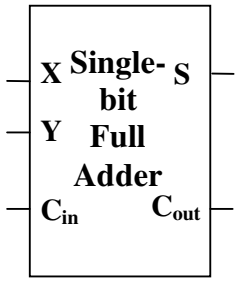
Q1	Q0	X	Z
0	**0**	**0**	**0**
0	**0**	**1**	**0**
0	**1**	**0**	**0**
0	**1**	**1**	**0**
1	**0**	**0**	**0**
1	**0**	**1**	**1**
1	**1**	**0**	**0**
1	**1**	**1**	**0**

$Z=Q_1Q_0'X$

In this solution the next state circuit is simpler.

Question 3– Part (a) -(7 pts):

Write a VHDL code to design a single-bit Full-Adder. You need to first draw the Truth table and/or write the logical expressions for the output signals before writing your VHDL code.



Solution:

a) Full Adder:

Truth Table is

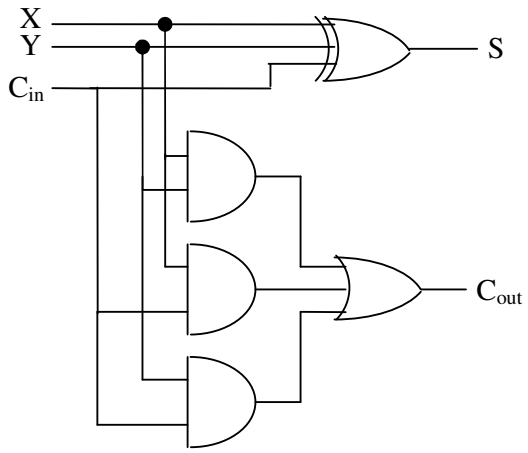
C_{in}	X	Y	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$S = C_{in}'(X \oplus Y) + C_{in}(X \oplus Y)' = X \oplus Y \oplus C_{in}$

To find a minimal function for C_{out} let us form a K-map

$Y \backslash C_{in}X$	00	01	11	10
0	0	0	1	0
1	0	1	1	1

$C_{out} = XY + XC_{in} + YC_{in}$

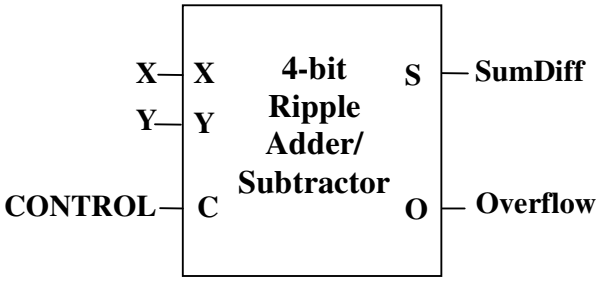


VHDL code:

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

entity fulladder is
    Port (X, Y, Cin: in BIT;
          Cout, S: out BIT);
end fulladder;

architecture Behavioral of fulladder is
begin
S<=X xor Y xor Cin;
Cout<= X and Y or X and Cin or Y and Cin;
end Behavioral;
```

Question 3– Part (b) -(15 pts):

Write a VHDL code to design a 4-bit 2's complement Ripple Adder/Subtractor using the Full-Adder module designed in part (a). The circuit receives two 4-bit 2's complement numbers and a control input depending on which it does addition or subtraction. CONTROL must be 0 to make addition and it must be 1 to make subtraction. The outputs are the sum/difference (SumDiff) and the overflow being set to 1 if overflow exists and 0 otherwise. You need to first draw the block diagram of the circuit then implement your design in VHDL.

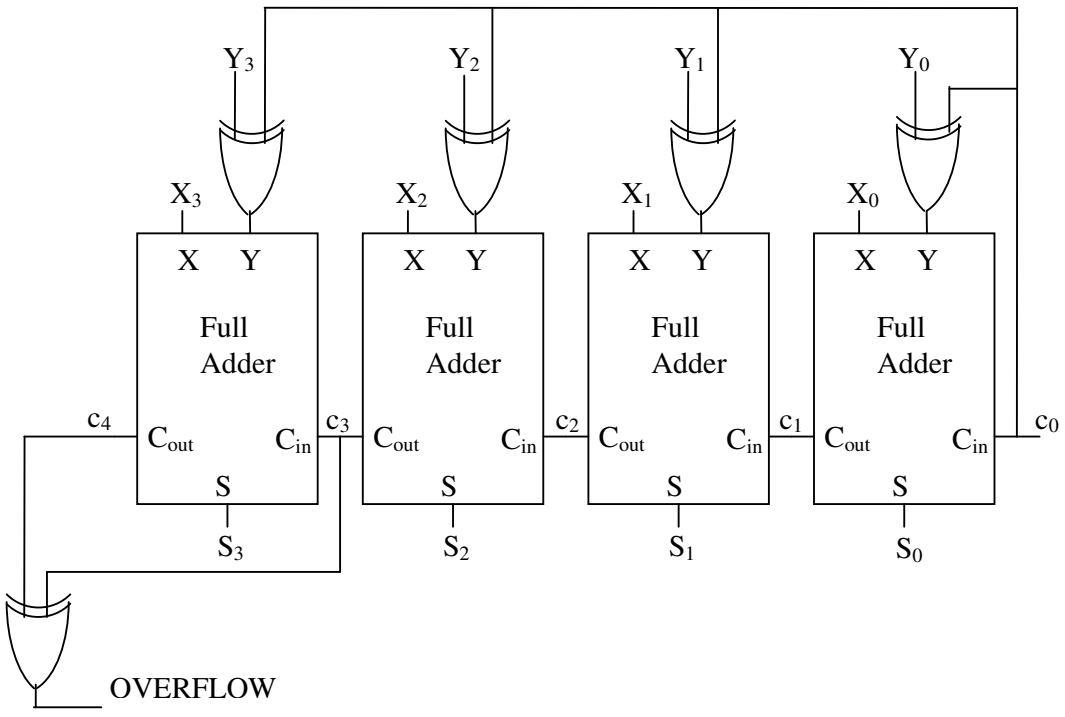


Solution:

b) Ripple Adder/Subtractor:

To perform addition of 2 4-bit numbers, we use 4 full adders and set the carry-in of the rightmost full adder to 0 (CONTROL = 0). There is overflow if $c_4 \neq c_3$. That is, OVERFLOW = 1 if $c_4 \neq c_3$ and OVERFLOW = 0 if $c_4 = c_3$.

For subtraction, we know that X-Y = X + (-Y) = X+Y$'$+1. Thus we complement Y and add 1 through setting c_0 to 1 (CONTROL = 1).



VHDL code:

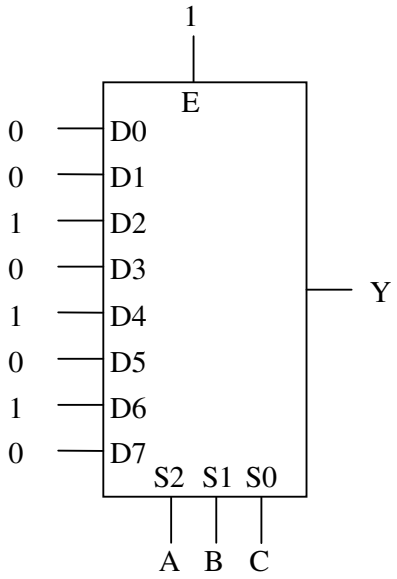
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity RippleAdder_Subtractor is
    Port(X: in BIT_VECTOR (3 downto 0);
         Y: in BIT_VECTOR (3 downto 0);
         CONTROL: in BIT;
         SumDiff: out BIT_VECTOR (3 downto 0);
         Overflow: out BIT);
end RippleAdder_Subtractor;

architecture Behavioral of RippleAdder_Subtractor is
component fulladder
    Port (X, Y, Cin: in BIT;
            Cout, S: out BIT);
end component;

signal C: BIT_VECTOR (4 downto 0);
signal Y_A: BIT_VECTOR (3 downto 0);

begin
C(0)<= CONTROL;
Y_A(0)<=CONTROL xor Y(0);
Y_A(1)<=CONTROL xor Y(1);
Y_A(2)<=CONTROL xor Y(2);
Y_A(3)<=CONTROL xor Y(3);

LABEL1: fulladder
    port map(X(0), Y_A(0), C(0), C(1), SumDiff(0));
LABEL2: fulladder
    port map(X(1), Y_A(1), C(1), C(2), SumDiff(1));
LABEL3: fulladder
    port map(X(2), Y_A(2), C(2), C(3), SumDiff(2));;
LABEL4: fulladder
    port map(X(3), Y_A(3), C(3), C(4), SumDiff(3));;

Overflow<=C(3) xor C(4);

end Behavioral;
```

Question 4-(22 pts):

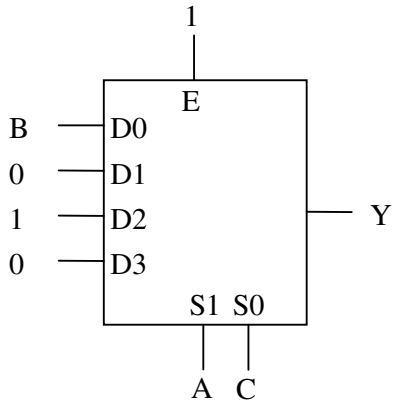
Implement F = (A+B)C'

a) (4 pts) using a generic 8-to-1 multiplexer and minimum number of additional simple gates,

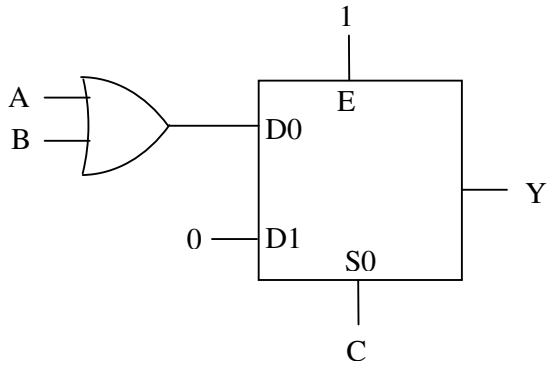
A	B	C	(A+B)C'
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0



b) (4 pts) using a generic 4-to-1 multiplexer and minimum number of additional simple gates,

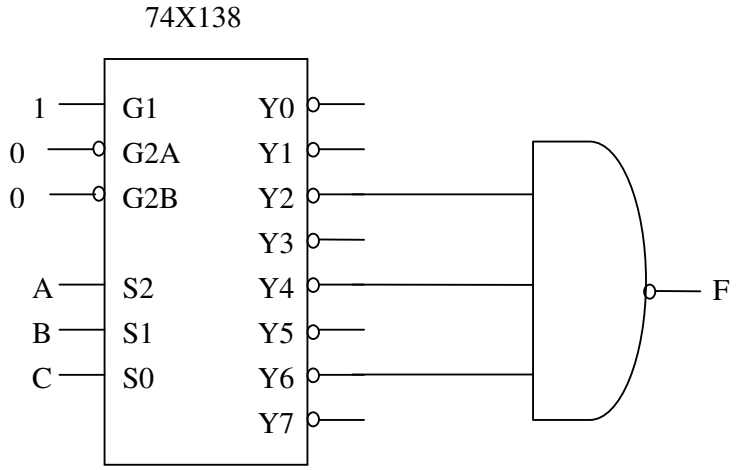


c) (4 pts) using a generic 2-to-1 multiplexer and minimum number of additional simple gates,

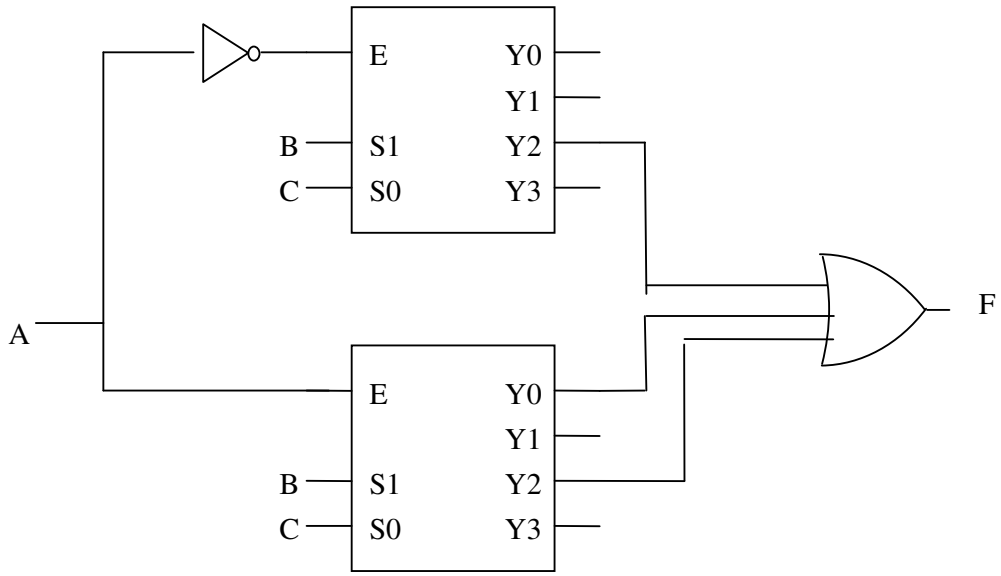


Question 4-(22 pts) - Continued:

d) (4 pts) using one 74XX138 decoder and minimum number of additional simple gates. (note that 74XX138 is a 3-to-8 binary decoder with active low outputs and with three enables, one active high and two active low),



e) (6 pts) using two 2-to-4 generic decoders and minimum number of additional simple gates.



Question 5-(22 pts):

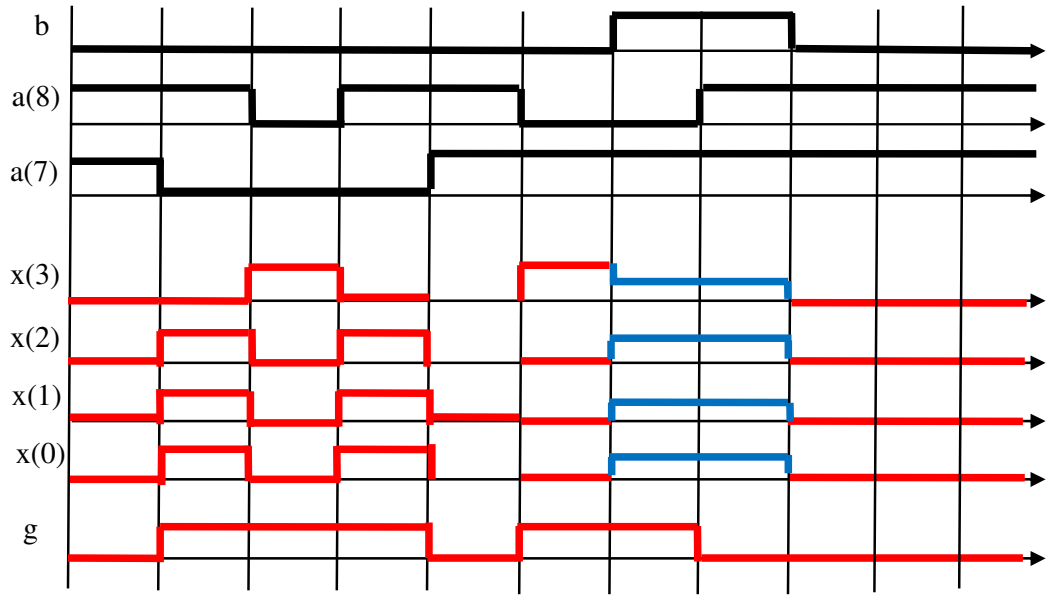
The following VHDL code is given:

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

entity top is
   Port ( b : in  BIT;
          a : in  BIT_VECTOR (9 downto 0);
          x : out  STD_LOGIC_VECTOR (3 downto 0);
          g : out  BIT);
end top;

architecture Behavioral of top is

component devre
   Port ( a : in  BIT_VECTOR (9 downto 0);
          d : out  STD_LOGIC_VECTOR (3 downto 0);
                              g : out BIT);
end component;
signal y: STD_LOGIC_VECTOR (3 downto 0);

begin
devre1: devre port map (a,y,g);
x<="ZZZZ" when b='1' else y;

end Behavioral;



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

entity devre is
   Port ( a : in  BIT_VECTOR (9 downto 0);
          d : out  STD_LOGIC_VECTOR (3 downto 0);
                              g : out BIT);
end devre;

architecture Behavioral of devre is

begin
              d<="0000" when (a="1111111111") else
                   "1001" when a(9)='0' else
                   "1000" when a(8)='0' else
                   "0111" when a(7)='0' else
                   "0110" when a(6)='0' else
                   "0101" when a(5)='0' else
                   "0100" when a(4)='0' else
```

```
                "0011" when a(3)='0' else
                "0010" when a(2)='0' else
                "0001" when a(1)='0' else
                "0000" when a(0)='0';
        g<='0' when (a="1111111111") else '1';
end Behavioral;
```

a) Complete the following timing diagram assuming that a(9)=a(6)=a(5)=a(4)=a(3)=a(2)=a(1)=a(0)=1.

b)



c) What is the function of the entity "devre"? How can you name it?
What is the function of the entity "top"? How can you name it?
What is the function of the signal "b"? How can you name it?
What is the function of the signal "g"? How can you name it?

Solution:

"devre" is a priority encoder with active low inputs giving highest priority to a(9).
"top" is the same thing but with tri-state encoding outputs and active low enable.
"b" is the active low enable of "top" . If b=1 then the output of "top" becomes high Z.
"g" is the active high got-something signal meaning that at least one of the inputs are low.

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.