15-12-2009

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

Surname: ________________________________

Name: ________________________________

ID-Number: ________________________________

Section Number: ________________________________

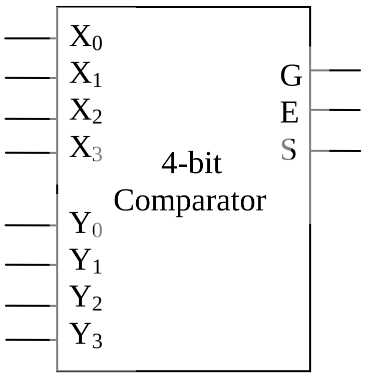
Signature: ________________________________

Duration is 120 minutes. Solve all 5 questions. Show all your work.
No books, notes, or calculators.

Q1 (20 points)	
Q2 (20 points)	
Q3 (20 points)	
Q4 (20 points)	
Q5 (20 points)	
Total	

Question 1-(20 pts):

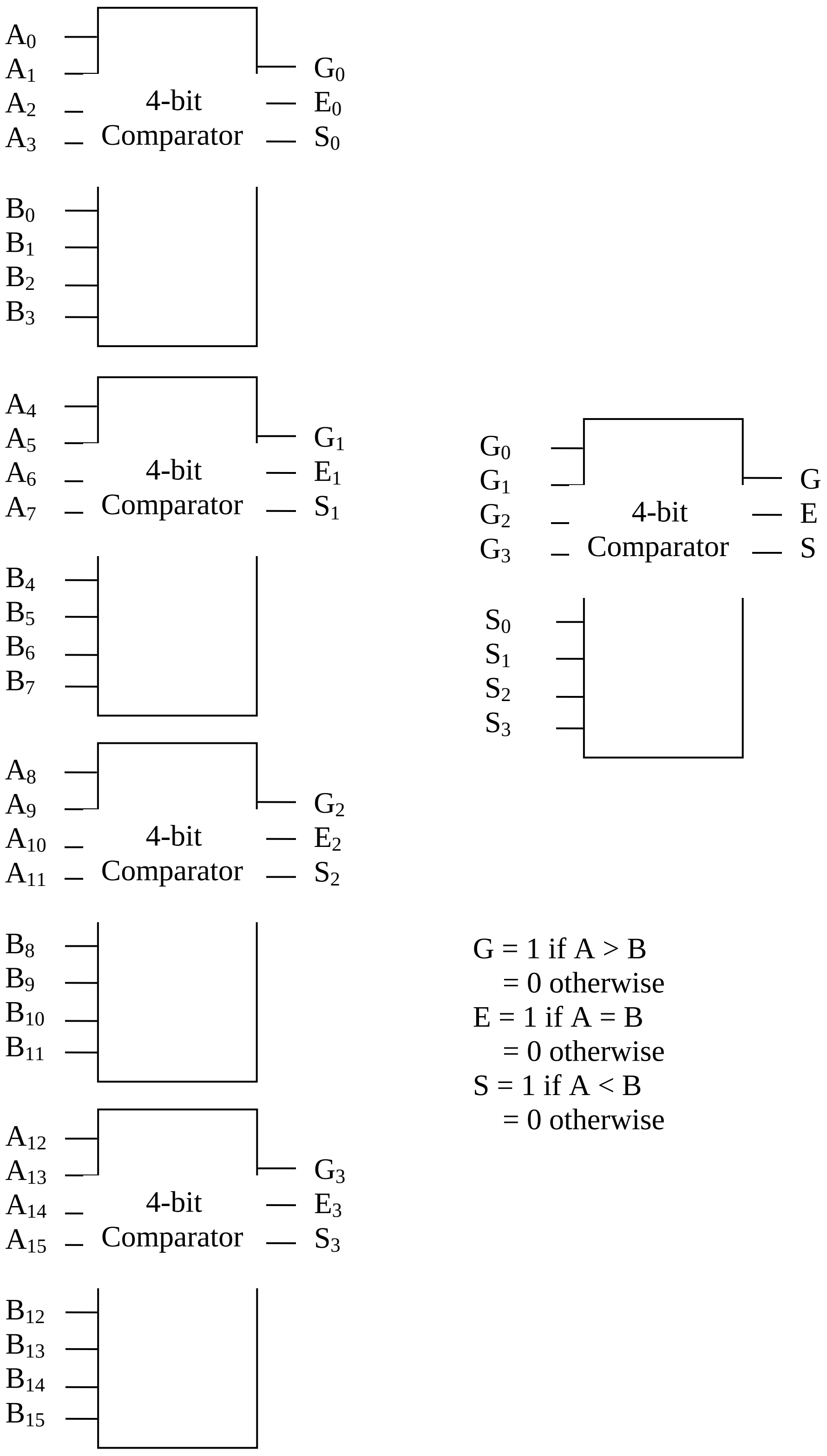
(a) (10 pts) The pin diagram and function description of a 4-bit comparator is given below.



G = 1 if X > Y
 = 0 otherwise
E = 1 if X = Y
 = 0 otherwise
S = 1 if X < Y
 = 0 otherwise

Use five of the above 4-bit comparator modules to design one 16-bit comparator.

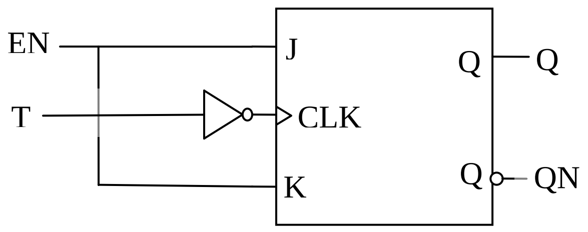
Solution:



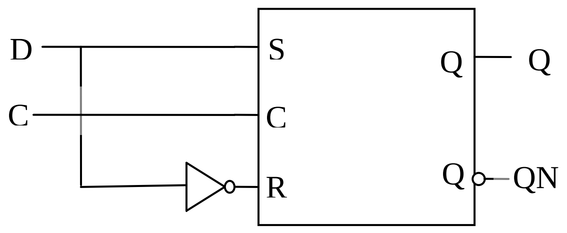
G = 1 if A > B
= 0 otherwise
E = 1 if A = B
= 0 otherwise
S = 1 if A < B
= 0 otherwise

Question 1-(20 pts) - Continued:

(b) (5 pts) Using a positive edge-triggered JK flip-flop and one or more additional gates, show how to implement a negative edge-triggered T flip-flop with Enable.
Solution:

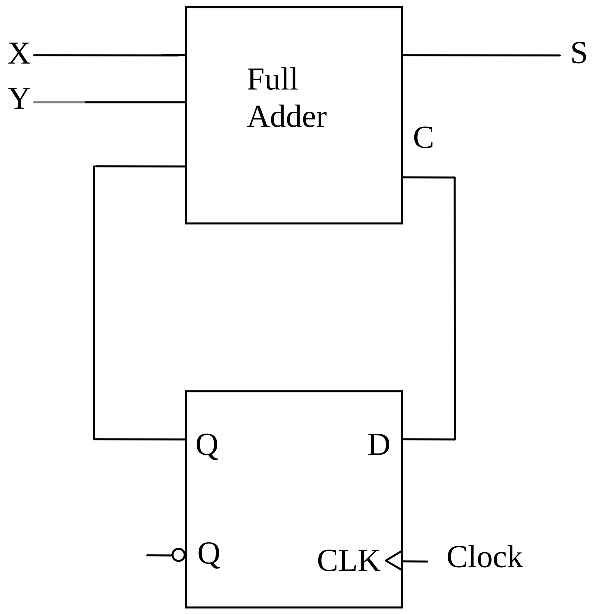


(c) (5 pts) Show how to implement a D latch using one or more additional simple gates and an SR latch with enable.
Solution:



Question 2-(20 pts):

A sequential circuit (FSM) has one flip-flop Q, two inputs X and Y, and one output S. It consists of a full-adder circuit connected to a D flip-flop, as shown below. Draw the state/output diagram, and write the next state table and output table of this sequential circuit. Assume that at Power ON Q is '0'. (Note: This circuit is a serial adder. S is equal to the sum of present X, present Y, and the carry at the last clock tick)



What is the value of Q between the 6th and 7th clock ticks, if the values of X and Y at the first 6 clock ticks are as given in the following table. Explain.

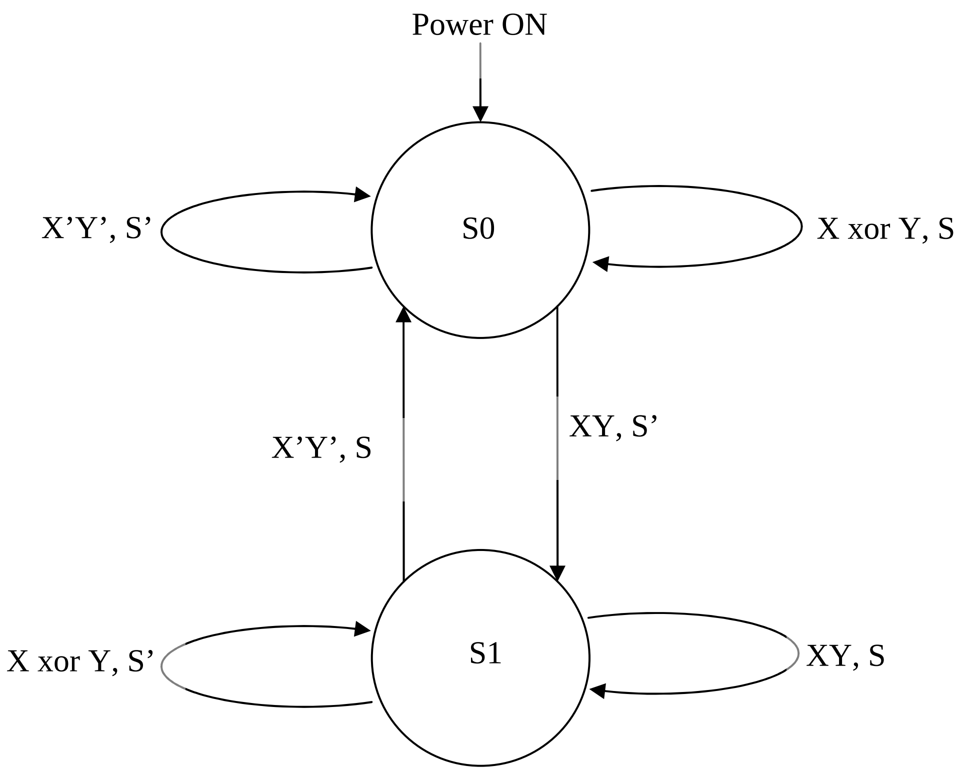
Clock tick	**1**	**2**	**3**	**4**	**5**	**6**
X	**0**	**0**	**1**	**0**	**1**	**1**
Y	**1**	**1**	**1**	**0**	**1**	**1**

Solution:

This is a serial adder which keeps the carry in its memory Q.

State of this FSM is the value of the carry. It has 2 states.
S0 = carry is '0'
S1 = carry is '1'
State encoding

State	Q
S0	0
S1	1



Next state and excitation table

Q	X (tick)	Y (tick)	Q* = D
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Optionally one can just write $D = XY + XQ + YQ$

Output table

Q	X (present)	Y (present	S
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Optionally one can just write $S = X \oplus Y \oplus Q$

Clock tick	**1**	**2**	**3**	**4**	**5**	**6**
X	**0**	**0**	**1**	**0**	**1**	**1**
Y	**1**	**1**	**1**	**0**	**1**	**1**
Q right after the tick (carry)	**0**	**0**	**1**	**0**	**1**	**1**

Therefore between 6th and 7th ticks Q = 1.

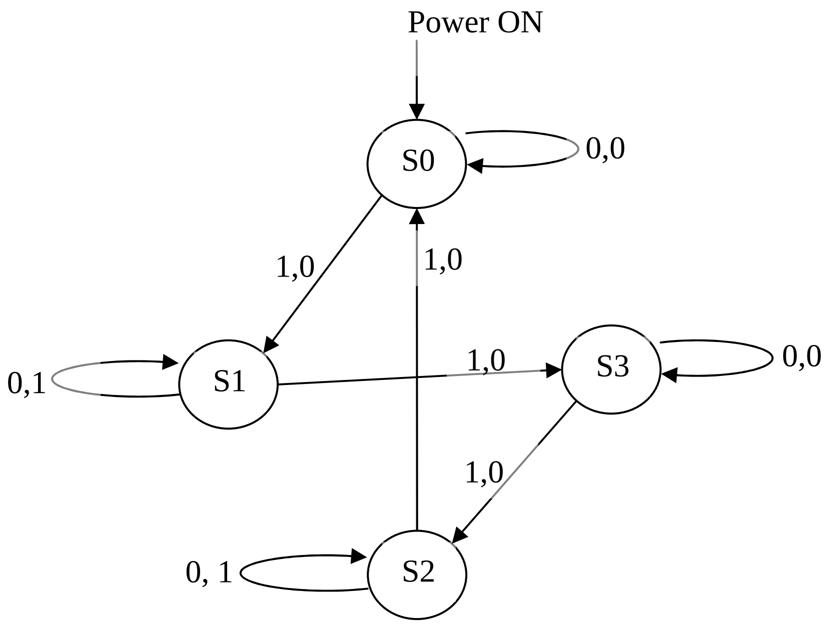
Question 3-(20 pts):

Design and draw a scnchronous FSM with two D flip-flops which has one input X, such that, when X = '0' the state of the FSM stays the same , and when X = '1' the FSM goes through the state transitions from "00" to "01", to "11", to "10", back to "00" and repeats. Output, F, is '1' if X = '0' and the state is "10", otherwise output is '0'. At Power ON the state should be "00". Is this FSM self-starting? Explain.
(Write state encoding, draw the state/output diagram, write next state table, excitation table, and output table, minimize the circuits, draw the circuit including initialization).

Solution:

State encoding

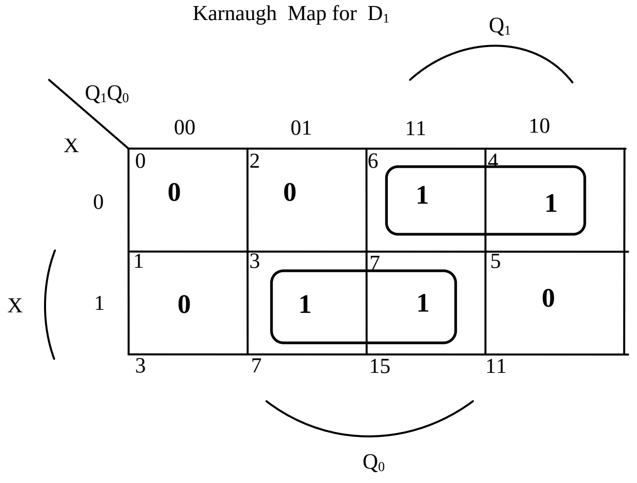
State	Q1	Q0
S0	0	0
S1	0	1
S2	1	0
S3	1	1



Next state and excitation table

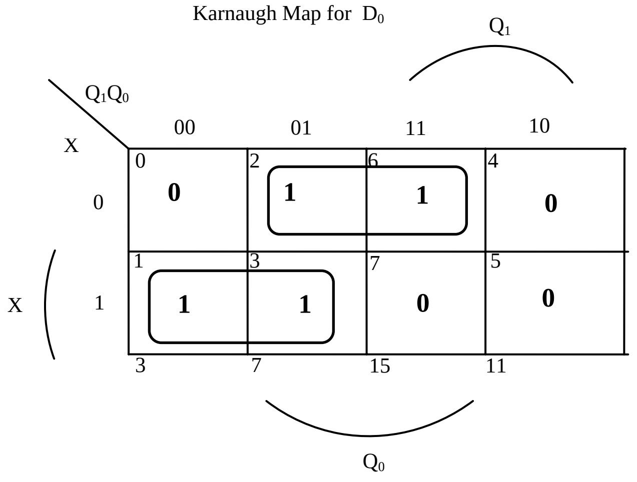
Q_1	Q_0	X (tick)	$Q_1^* = D_1$	$Q_0^* = D_0$
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	1
1	0	0	1	0
1	0	1	0	0
1	1	0	1	1
1	1	1	1	0

Karnaugh Map for D_1



$D_1 = Q_1X' + Q_0X$

Karnaugh Map for D_0



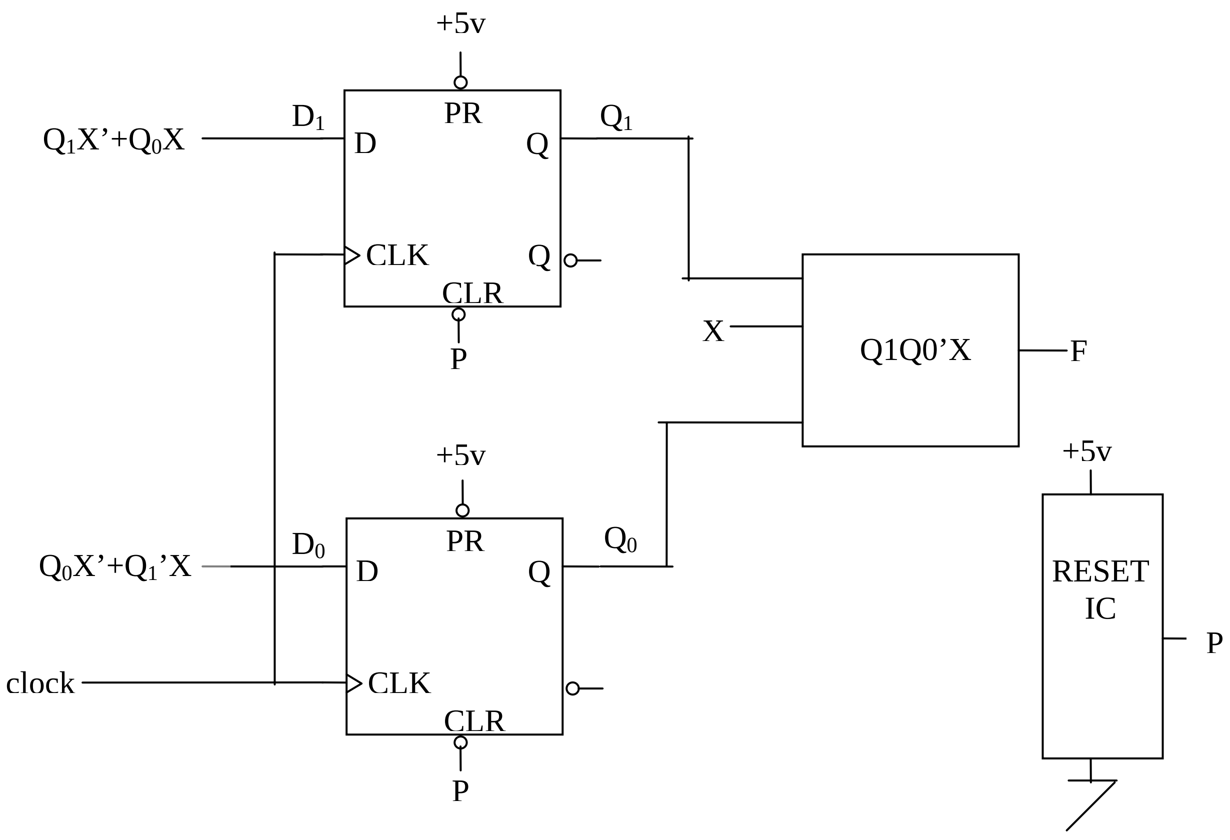
$D_0 = Q_0X' + Q_1'X$

Output table

Q_1	Q_0	X (present)	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

No need for a karnaugh map. $F = Q_1Q_0'X$

Circuit including initialization



Question 4 (20 pts):

Write VHDL code to implement the 74x148-like 8-input piriority encoder. The entity is provided to you below. All of the signals have "_L" indicating their active-low operations.

```
entity V74x148 is
    port( E_L: in STD_LOGIC;
             I_L: in STD_LOGIC_VECTOR(7 downto 0);
             A_L: out STD_LOGIC_VECTOR(2 downto 0);
             EO_L: out STD_LOGIC;
             GS_L: out STD_LOGIC );
end V74x148;
```

-- *Write the architecture of your code below* --

Solution:

-- The architecture is provided on page 416 of the textbook.--

```
architecture V74x148p of V74x148 is
    signal EI: std_logic;                        -- active-high version of input
    signal I: std_logic_vector(7 downto 0);  -- active-high version of inputs
    signal EO, GS: std_logic;                  -- active-high version of outputs
    signal A: std_logic_vector(2 downto 0); -- active-high version of outputs
begin
    process (EI_L, I_L, EI, EO, GS, I, A)
    variable j: INTEGER range 7 downto 0;
    begin
             EI <= not EI_L;     -- convert inputs
             I <= not I_L;         -- convert inputs
             EO <= '1' ; GS<= '0'; A<= "000";
             if (EI) = '0' then EO <= '0';
             else for j in 7 downto 0 loop
                    if I(j) = '1' then
                        GS <= '1'; EO <= '0'; A<= CONV_STD_LOGIC_VECTOR(j,3);
                         exit;
                    endif;
               end loop;
             end if;
             EO_L <= not EO;  -- convert output
             GS_L <= not GS;  -- convert output
             A_L <= not A;       -- convert outputs
      end process;
end V74x148p;
```

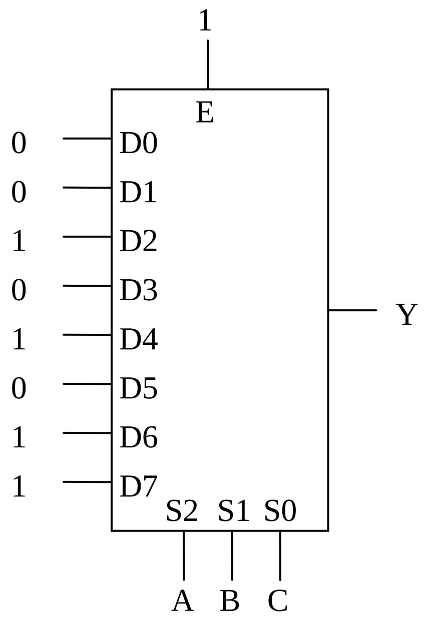
Question 5 (20 pts):

Implement F = (A⊕B).C’ + AB

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

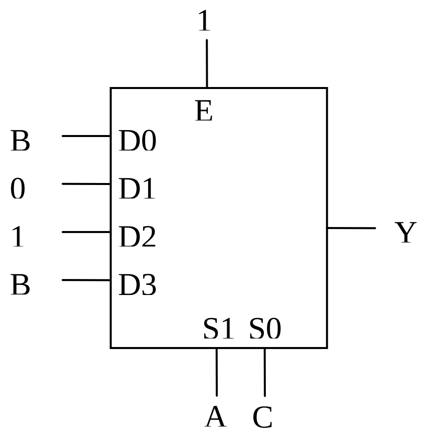
Solution:

A	B	C	(A⊕B).C’ + AB
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	1



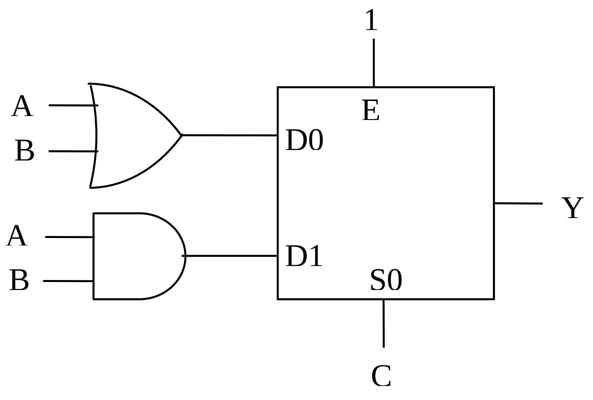
b) (4 pts) using a generic 4-to-1 multiplexer and minimum number of additional simple gates,

Solution:



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

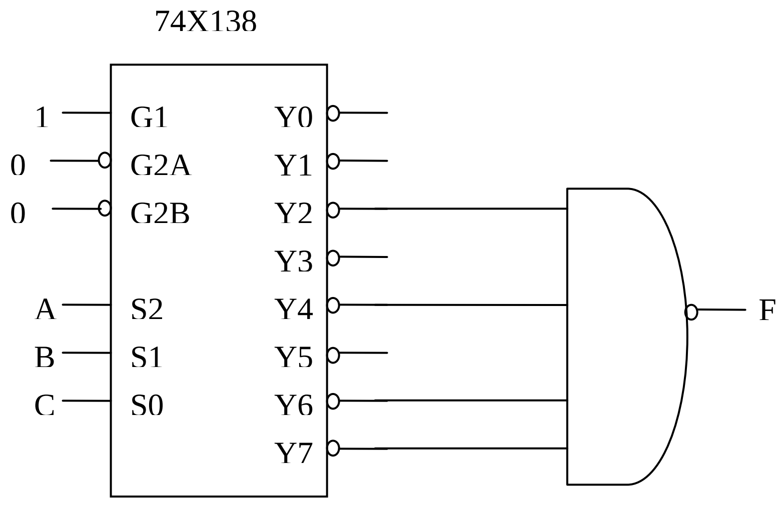
Solution:



Question 5-(20 pts) - Continued:

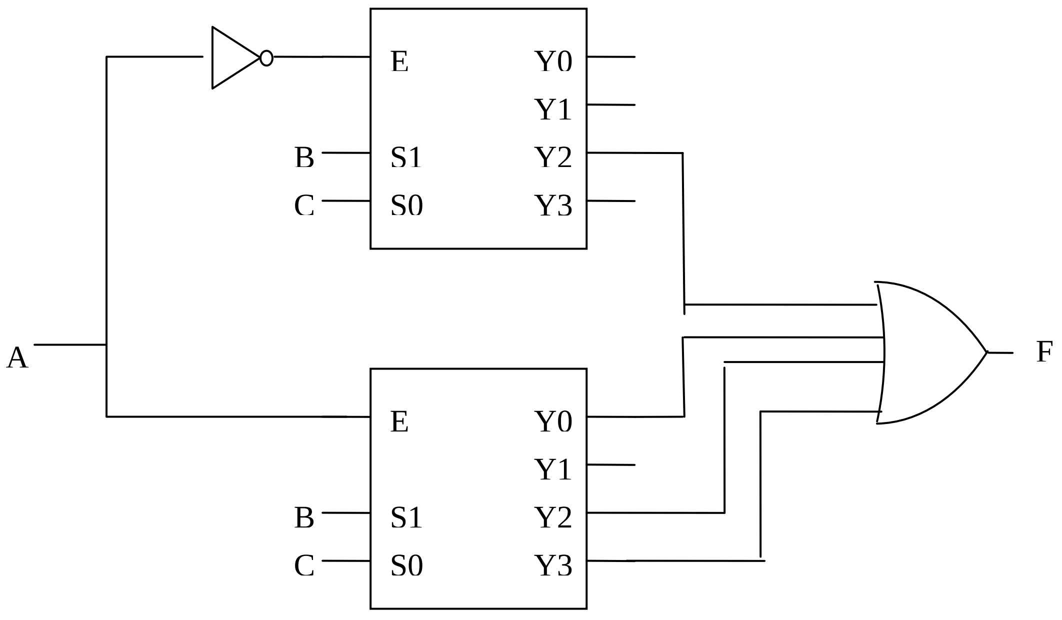
d) (4 pts) using one 74X138 decoder and minimum number of additional simple gates. (note that 74X138 is a 3-to-8 binary decoder with active low outputs and with three enables, one active high and two active low),

Solution:



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

Solution:



Appendix: Some VHDL Templates 15-12-2009

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.