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: _____

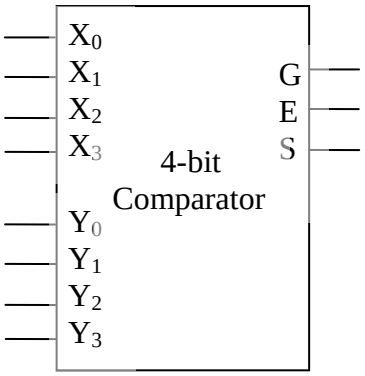
Signature: _____

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):

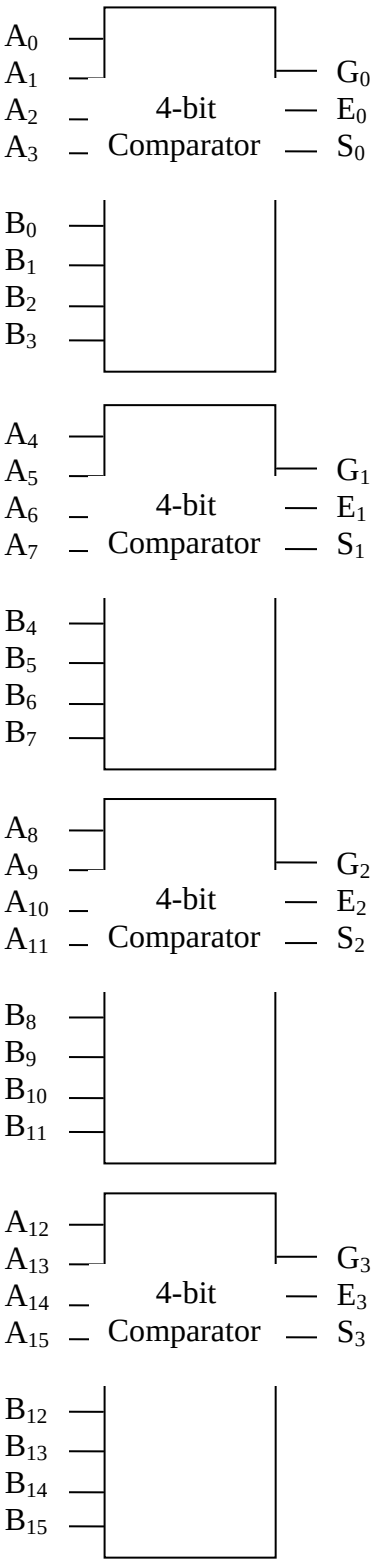
(a) (10 pts) The pin diagram and function description of a 4-bit comparator is given below.

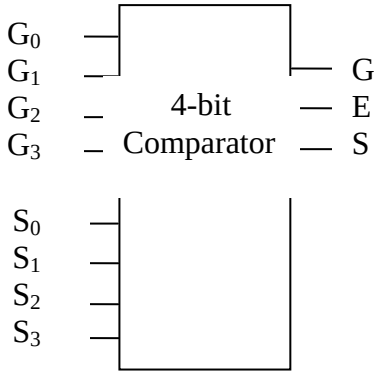


G = 1 if X > Y
= 0 otherwise
E = 1 if X = Y
= 0 otherwise
S = 1 if X < Y
= 0 otherwise

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

Solution:

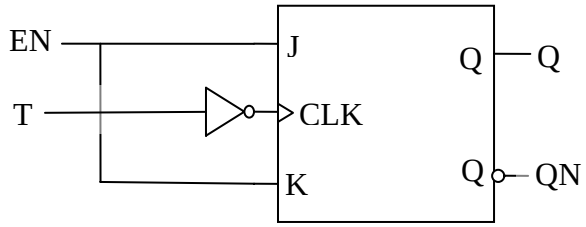




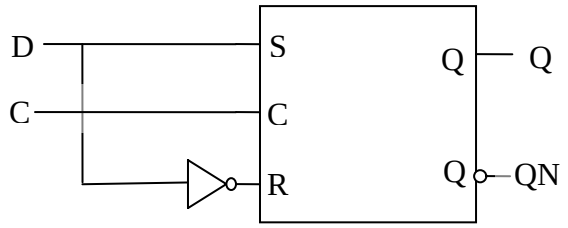
G = 1 if A > B
= 0 otherwise
E = 1 if A = B
= 0 otherwise
S = 1 if A < B
= 0 otherwise

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:

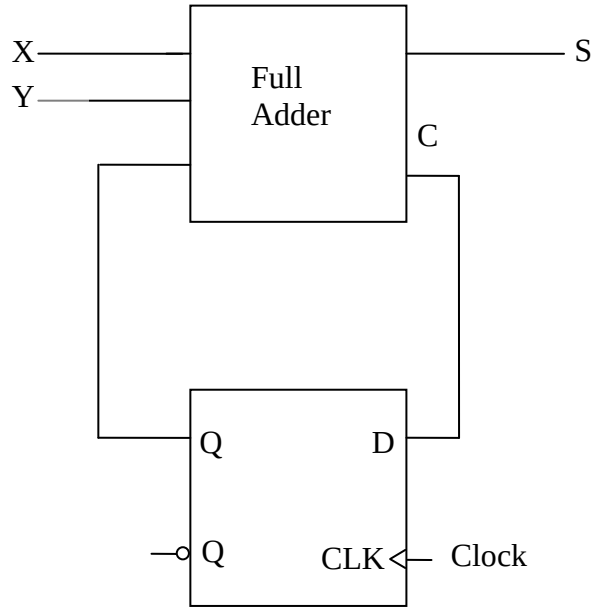


(c) (5 pts) Show how to implement a D latch using one or more additional simple gates and an SR latch with enable.
Solution:



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.

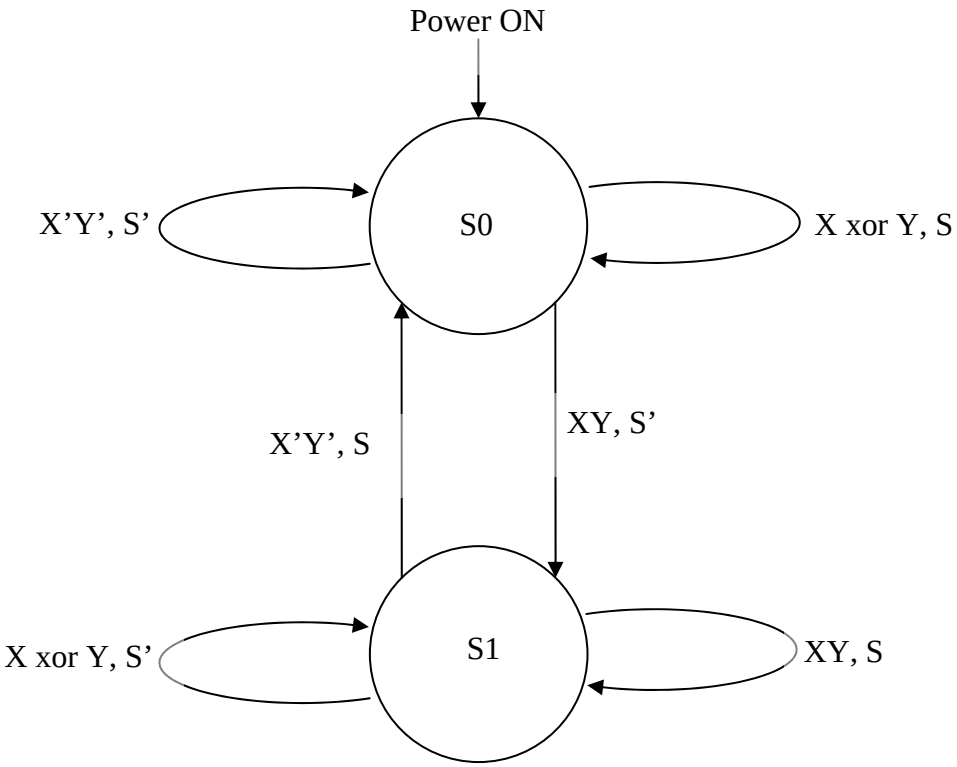
Clock tick	**1**	**2**	**3**	**4**	**5**	**6**
X	**0**	**0**	**1**	**0**	**1**	**1**
Y	**1**	**1**	**1**	**0**	**1**	**1**

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 6^{th} and 7^{th} ticks Q = 1.

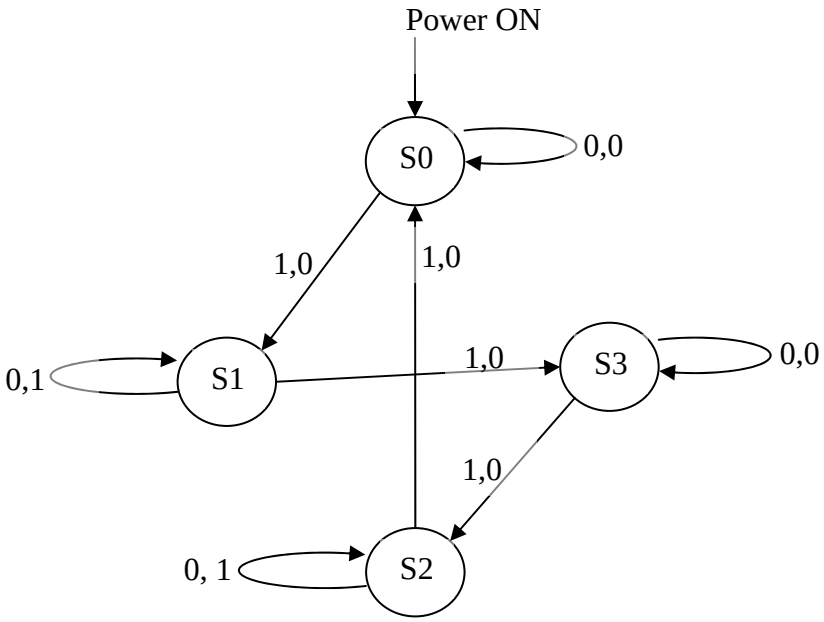
Question 3-(20 pts):

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

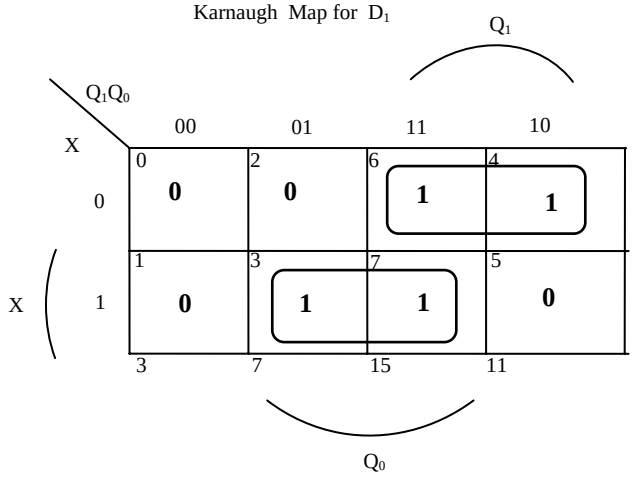
State	Q1	Q0
S0	0	0
S1	0	1
S2	1	0
S3	1	1



Next state and excitation table

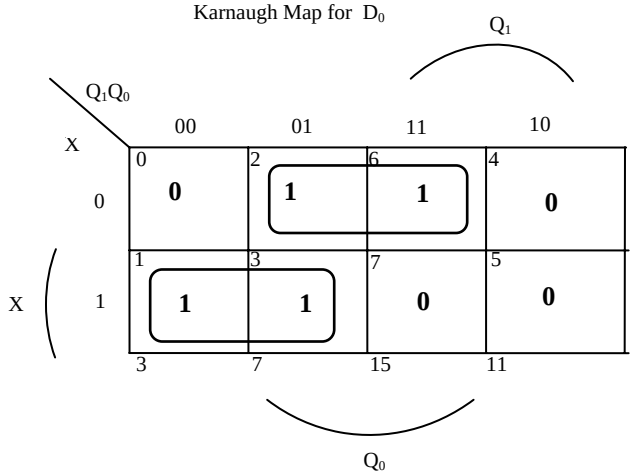
Q_1	Q_0	X (tick)	$Q_1^* = D_1$	$Q_0^* = D_0$
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	1
1	0	0	1	0
1	0	1	0	0
1	1	0	1	1
1	1	1	1	0

Karnaugh Map for D_1



$D_1 = Q_1X' + Q_0X$

Karnaugh Map for D_0



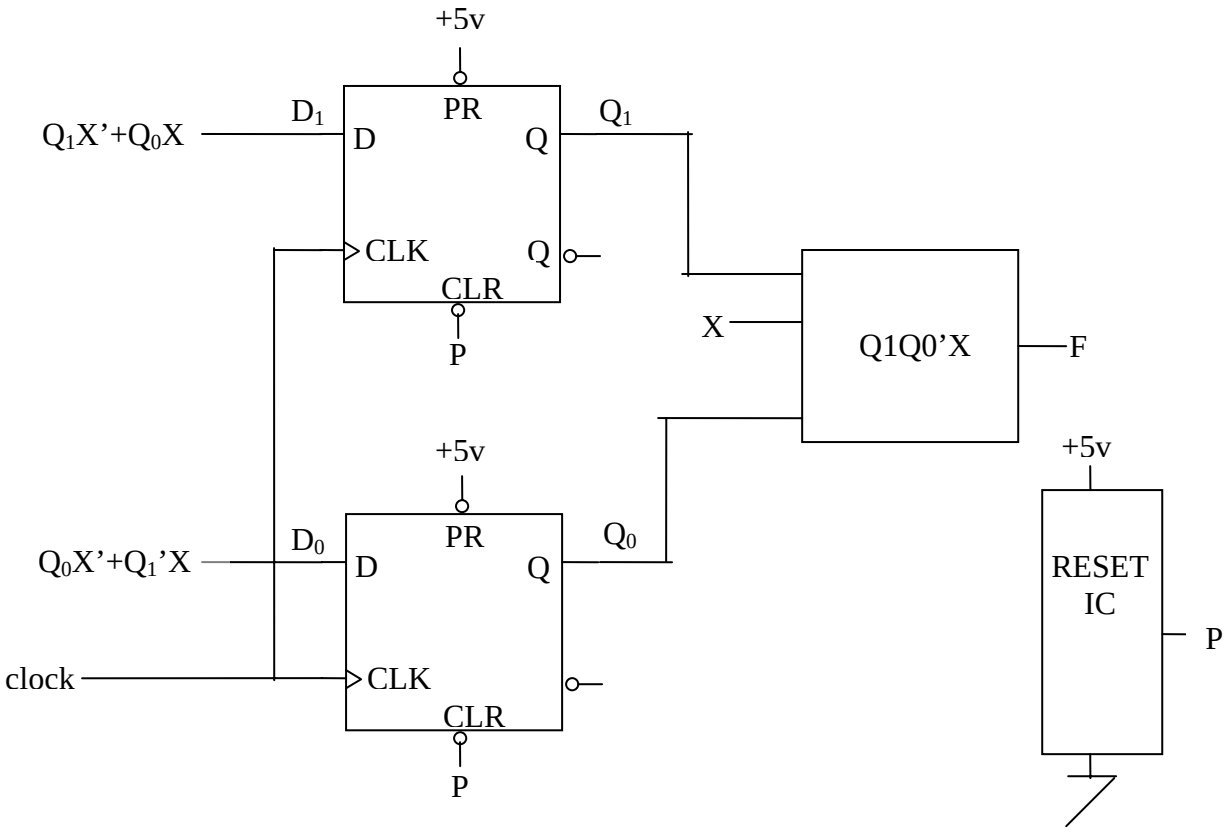
$D_0 = Q_0X' + Q_1'X$

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;
```

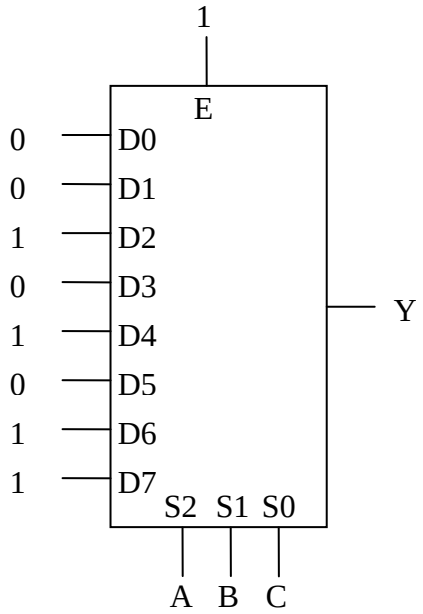
Question 5 (20 pts):

Implement F = (A⊕B).C' + AB

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

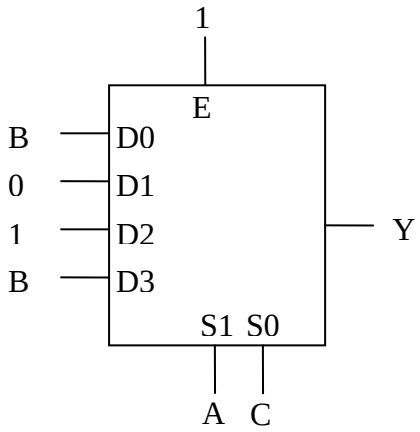
Solution:

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



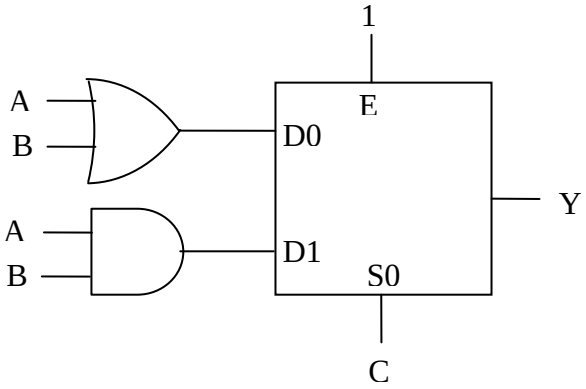
b) (4 pts) using a generic 4-to-1 multiplexer and minimum number of additional simple gates,

Solution:



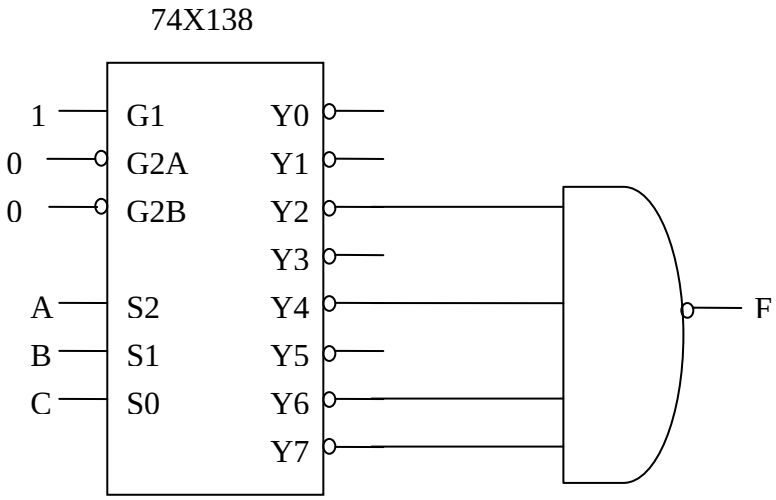
c) (4 pts) using a generic 2-to-1 multiplexer and minimum number of additional simple gates,

Solution:

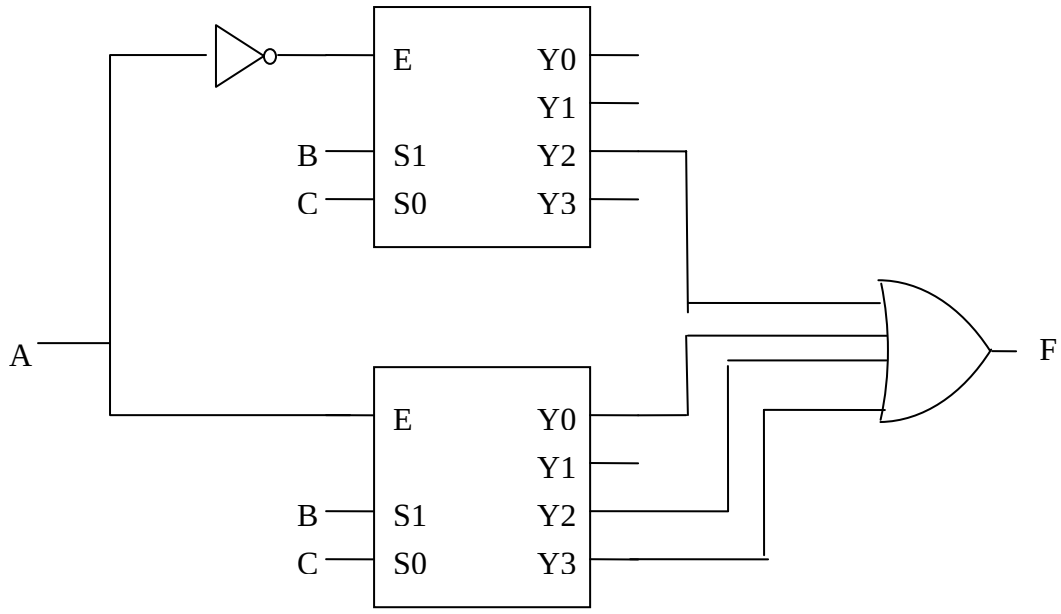


Question 5-(20 pts) - Continued:

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.

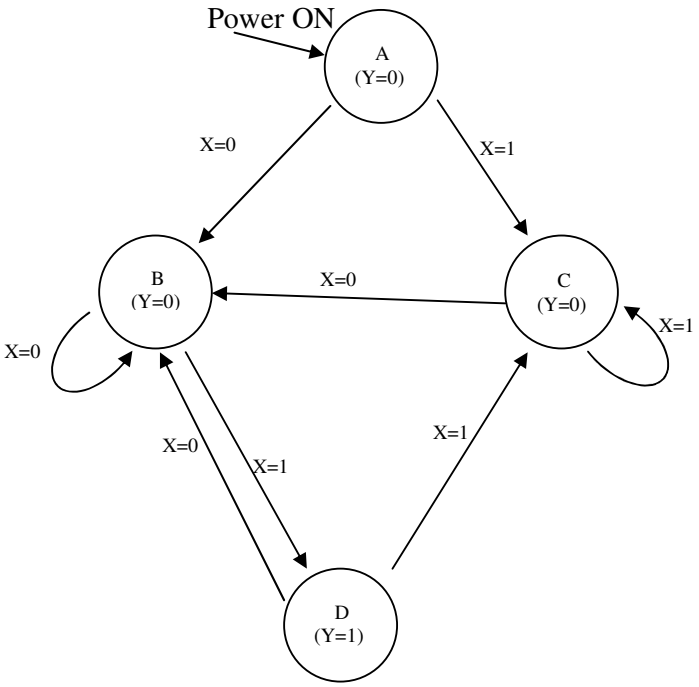
4-5-2008
BILKENT UNIVERSITY
Department of Electrical and Electronics Engineering
EEE102 Introduction to Digital Circuit Design
Midterm Exam II SOLUTION

Surname: _____
Name: _____
ID-Number: _____
Signature: _____

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

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

Q1. A synchronous FSM, with one input X and one output Y, has 4 states A, B, C, and D and has the following State/Output diagram. Design and draw this FSM using D flip flops with active low Preset and Clear controls. What does this FSM do?



Solution:

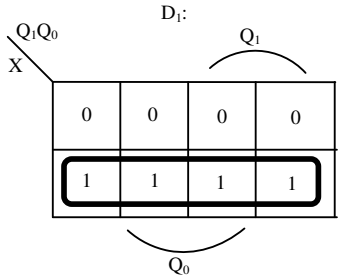
State encoding

State	Q_1	Q_0
A	0	0
B	0	1
C	1	0
D	1	1

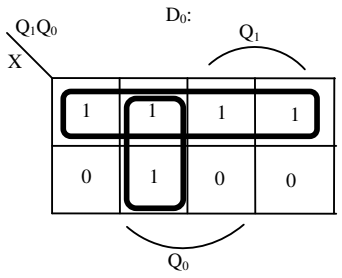
Next state and excitation table

Q_1	Q_0	X	Q_1*=D_1	Q_0*=D_0
0	0	0	0	1
0	0	1	1	0
0	1	0	0	1
0	1	1	1	1
1	0	0	0	1
1	0	1	1	0
1	1	0	0	1
1	1	1	1	0

Minimized functions for the next state logic



$D_1 = X$

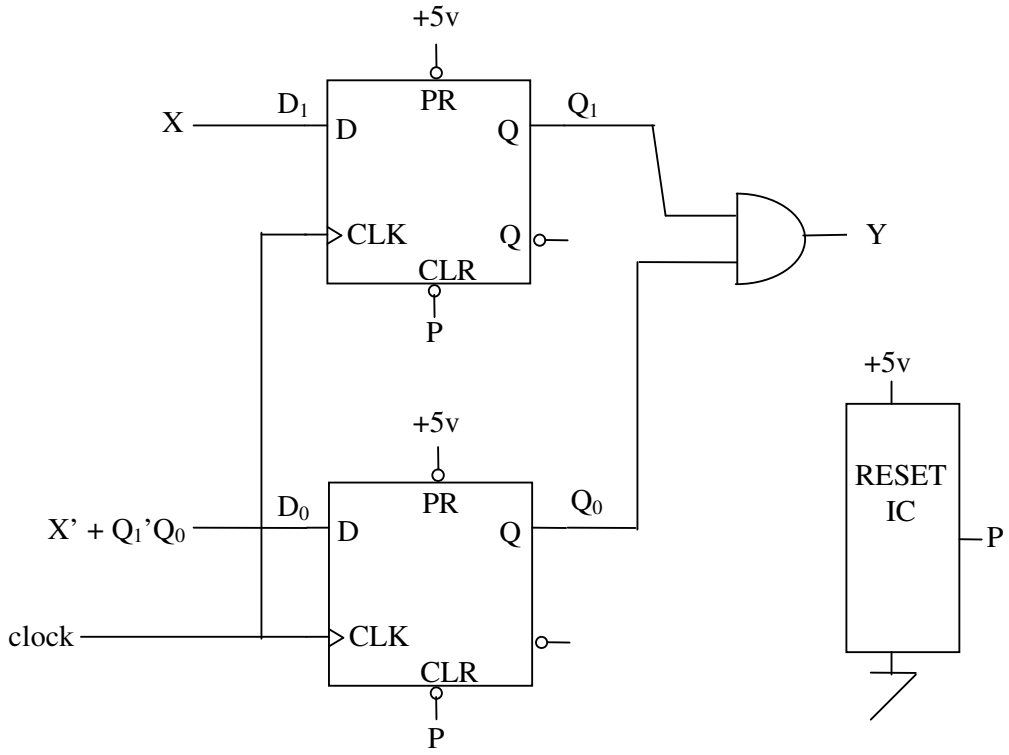


$D_0 = X' + Q_1'Q_0$

Output table

Q_1	Q_0	Y
0	0	0
0	1	0
1	0	0
1	1	1

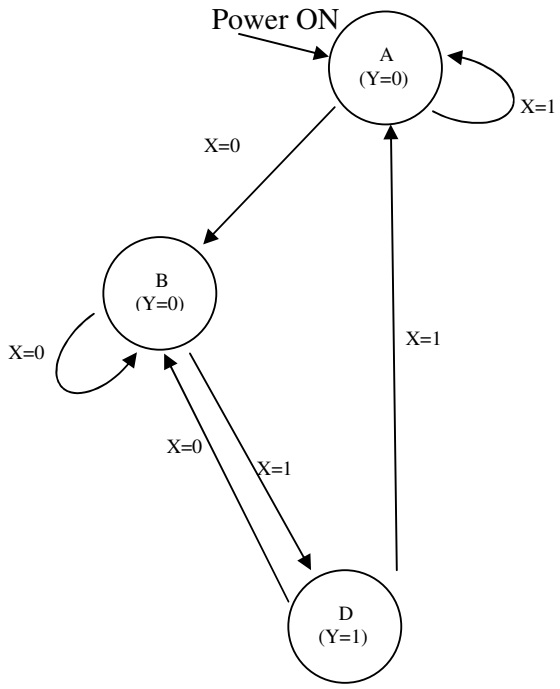
$Y = Q_1Q_0$



This FSM makes its output 1 when the last received X is 1 and the previously received X is 0. Thus it detects the "01" sequence in the input.

Alternative solution:

It is observed that states A and C are equivalent. Therefore we can draw the State/Output diagram again.



State	Q_1	Q_0
A	0	0
B	0	1
D	1	0

Q_1	Q_0	X	Q_1*=D_1	Q_0*=D_0
0	0	0	0	1
0	0	1	0	0
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	d	d
1	1	1	d	d

Q_1	Q_0	Y
0	0	0
0	1	0
1	0	1
1	1	d

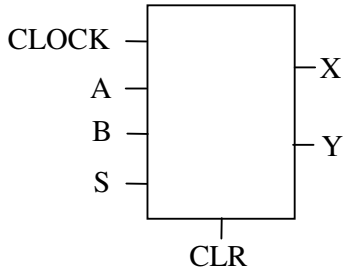
Minimized functions: $D_1 = Q_0X$, $D_0 = X$, $Y = Q_1$

Q2. A circuit, shown below as a block, realizes the following functions:

(i) At the rising edge of the CLOCK signal
it loads A to X and B to Y if S = 0, and
it loads B to X and A to Y if S = 1.

(ii) If the asynchronous CLR input is 1 then it makes X = 0 and Y = 0.

"Asynchronous" means "independent of CLOCK edges". Write VHDL code to describe this circuit.



Solution:

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

entity AdvancedDff is
   Port ( CLOCK : in std_logic;
          CLR : in std_logic;
          A : in std_logic;
          B : in std_logic;
```

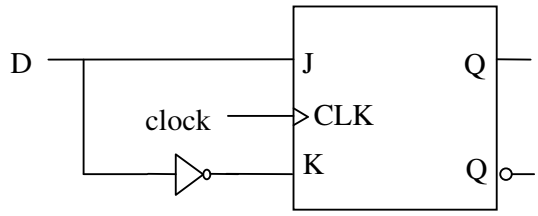
```
        S : in std_logic;
        X : out std_logic;
        Y : out std_logic);
end AdvancedDff;

architecture Behavioral of AdvancedDff is
begin
process (CLOCK,CLR)
        begin
                if CLR='1' then X<='0';Y<='0';
                elsif CLOCK'event and CLOCK='1' then
                        case S is
                                when '0' => X<=A;Y<=B;
                                when '1' =>     X<=B;Y<=A;
                                when others => X<=A;Y<=B;
                        end case;
                end if;
end process;
end Behavioral;
```

Q3. a) Draw how you can obtain a D flip flop using a single JK flip flop and minimum number of additional simple gates.

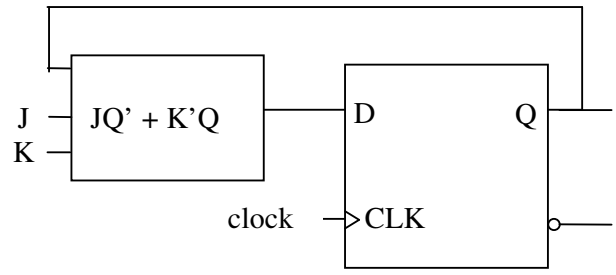
b) Draw how you can obtain a JK flip flop using a single D flip flop and minimum number of additional simple gates.

Solution:

a)



b)



Q4. Implement F = AB + C'

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

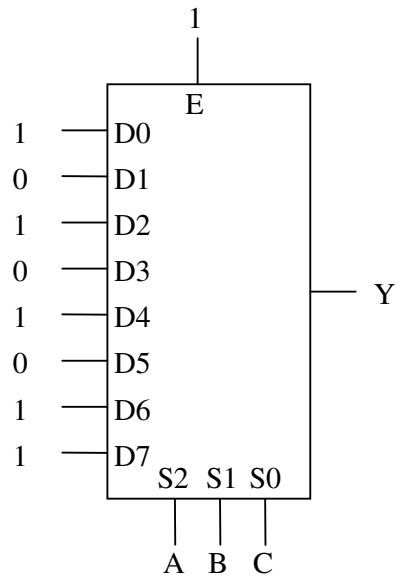
b) using a generic 4-to-1 multiplexer and minimum number of additional simple gates,

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

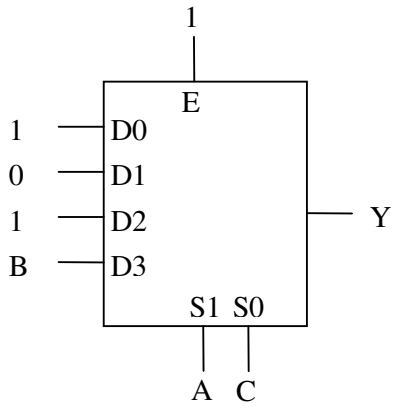
d) using one 74XX138 decoder and minimum number of additional simple gates. (note that 74XX138 is a 3-to-8 binary decoder with active low outputs and with three enables, one active high and two active low),
and
e) using two 2-to-4 generic decoders and minimum number of additional simple gates.
Solution:

A	B	C	AB+C'
0	0	0	1
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

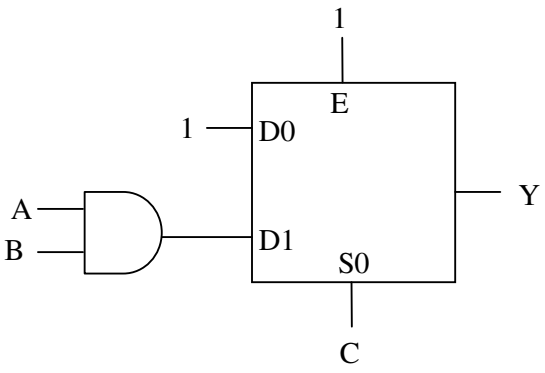
a) using a generic 8-to-1 multiplexer



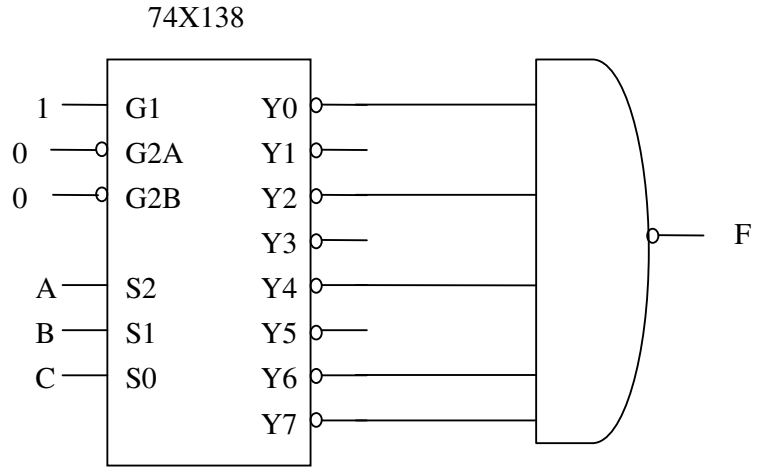
b) using a generic 4-to-1 multiplexer



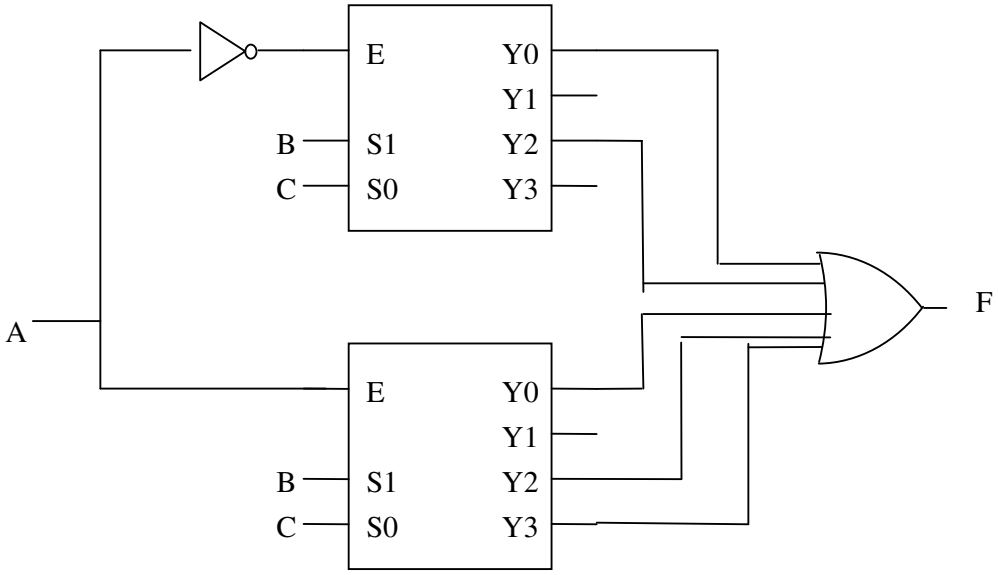
c) using a generic 2-to-1 multiplexer



d) using a 74XX138 decoder.



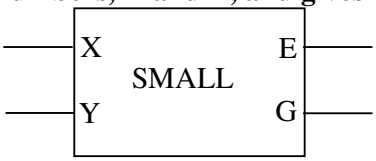
e) using two 2-to-4 generic decoders.



Q5. We need to design a comparator (called BIG) which has two 2-bit binary numbers as input, A and B, and two outputs EQ (meaning "equal") and GT (meaning "greater") such that the following table is implemented:

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

We have already designed a simple comparator (called SMALL) which receives two 1-bit numbers, X and Y, and gives outputs E and G such that



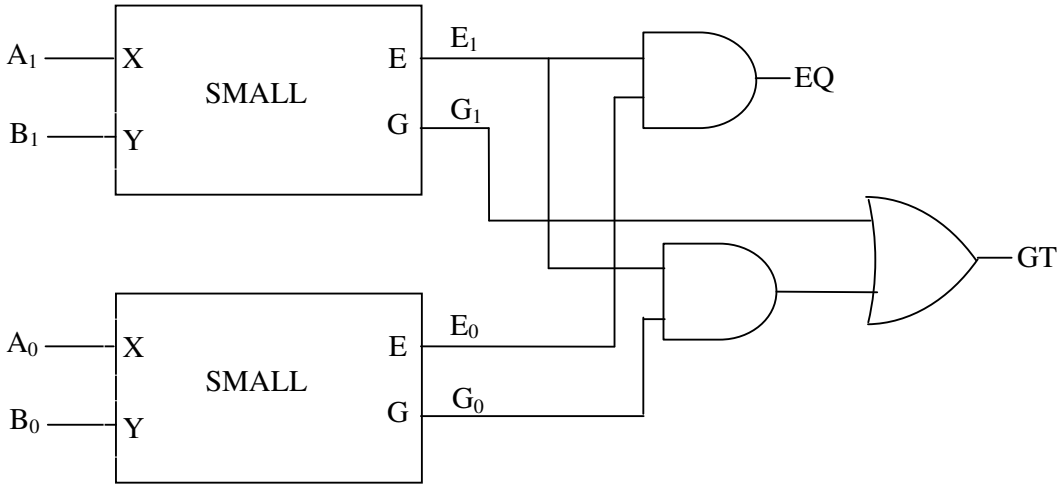
Condition	E	G
X = Y	1	0
X > Y	0	1
X < Y	0	0

a) (5 points) Use two SMALLs and minimum number of additonal simple gates to design one BIG. Draw your circuit.

b) (15 points) Assume now that the VHDL code for SMALL has already been written and included in the library. Write VHDL code for BIG using two SMALLs as components.

Solution:

a)



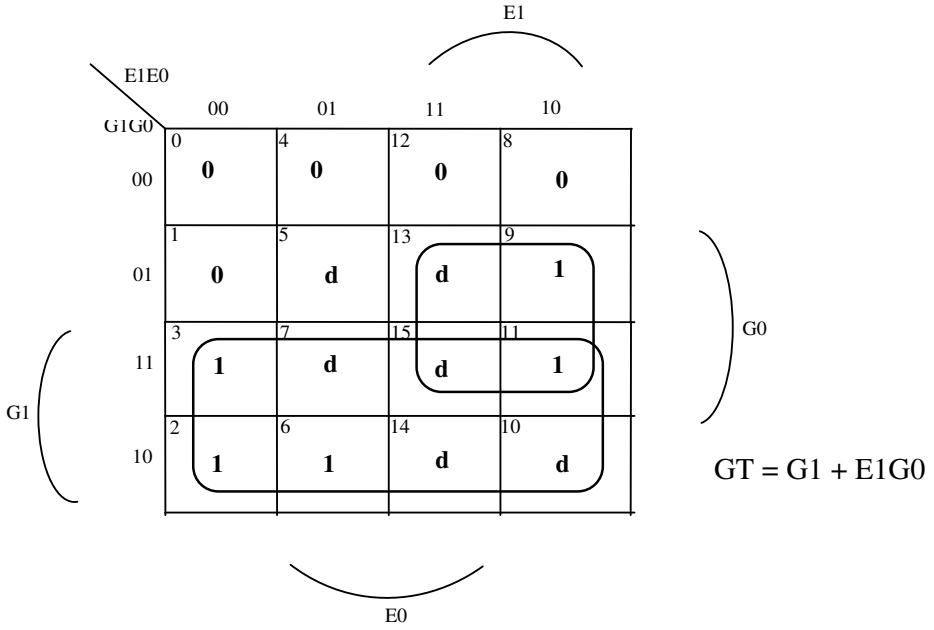
The logic behind this solution is as follows: EQ is 1 if and only if $A_1 = B_1$ and $A_0 = B_0$.
GT is 1 if $A_1 > B_1$ (in this case we do not have to compare A_0 and B_0),
or $A_1 = B_1$ and $A_0 > B_0$.
Thus $GT = G_1 + E_1G_0$

More systematic solution:

A1	B1	A0	B0		E1	E0	G1	G0		GT
0	0	0	0		1	1	0	0		0
0	0	0	1		1	0	0	0		0
0	0	1	0		1	0	0	1		1
0	0	1	1		1	1	0	0		0
0	1	0	0		0	1	0	0		0
0	1	0	1		0	0	0	0		0

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

E1	E0	G1	G0		GT
0	0	0	0		0
0	0	0	1		0
0	0	1	0		1
0	0	1	1		1
0	1	0	0		0
0	1	0	1		d
0	1	1	0		1
0	1	1	1		d
1	0	0	0		0
1	0	0	1		1
1	0	1	0		d
1	0	1	1		1
1	1	0	0		0
1	1	0	1		d
1	1	1	0		d
1	1	1	1		d



GT = G1 + E1G0

b)

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

entity BIG is
   Port ( A : in std_logic_vector(1 downto 0);
          B : in std_logic_vector(1 downto 0);
          EQ : out std_logic;
          GT : out std_logic);
end BIG;

architecture Behavioral of BIG is
component SMALL
Port ( X : in std_logic;
       Y : in std_logic;
       E : out std_logic;
       G : out std_logic);
end component;
signal E1,E0,G1,G0:std_logic;
begin
C1:SMALL port map(A(1),B(1),E1,G1);
C2:SMALL port map(A(0),B(0),E0,G0);
EQ <= E1 and E0;
GT <= G1 or (E1 and G0);
end Behavioral;
```

You do not have to write the code for SMALL but I have included it below for your information:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity SMALL is
   Port ( X : in std_logic;
          Y : in std_logic;
          E : out std_logic;
          G : out std_logic);
end SMALL;
architecture Behavioral of SMALL is
begin
process(X,Y)
begin
          if X=Y then E<='1';G<='0';
          elsif X>Y then E<='0';G<='1';
          else E<='0';G<='0';
          end if;
end process;
end Behavioral;
```

Lastname, Name:_____
ID:_____

EEE 102: Digital Systems Design

Midterm Exam 2

April 27, 2013

Duration: 90 min

Q	1	2	3	Total
Pts	30	35	35	100
Score				

This is a closed-book and closed-notes exam.

NO CREDIT will be given to answers without clear, formal and clean

JUSTIFICATION.

1. **[30 pts]** Design a special 4-bit register with two 1-bit inputs $S1$ and $S2$. This clocked register performs the following operations depending on its inputs:

S1	S2	Operation
0	0	Circular shift left 1-bit
0	1	Circular shift right 1-bit
1	0	Parallel load
1	1	Idle

2. **[35 pts]**

Design a sequential circuit that outputs 1 when the total number of 1's on the input (including the current input) is a multiple of 3. Otherwise the circuit outputs 0. As an example,

x	0	1	1	0	1	0	1	...
y	0	0	0	0	1	1	0	...

where x is the input and y is the output.

You can only use MB-type flip-flop, where an MB type flip-flop is defined as follows:

M	B	Q(t+1)
0	0	1
0	1	1
1	0	0
1	1	Q(t)

and a minimum number of NAND gates.

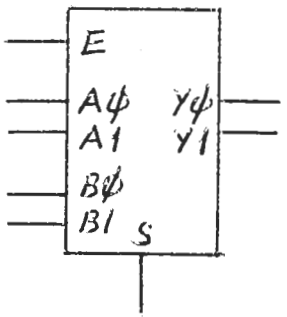
3. **[35 pts]** Design an integer divider using an ASM chart. Your machine starts with loading two binary numbers: A in the register $R1$ and B in the register $R2$. Then, it calculates $\lfloor A/B \rfloor$ and stores the result in the register $R3$. Here, $\lfloor A/B \rfloor$ represents the integer number of B's in A. As an example if $A = 5$ and $B = 2$, then the result is $\lfloor A/B \rfloor = 2$ stored in $R3$.

   a. [5 pts] Give the algorithm in a pseudo-code form.

   b. [15 pts] Built the ASM chart for your divider.

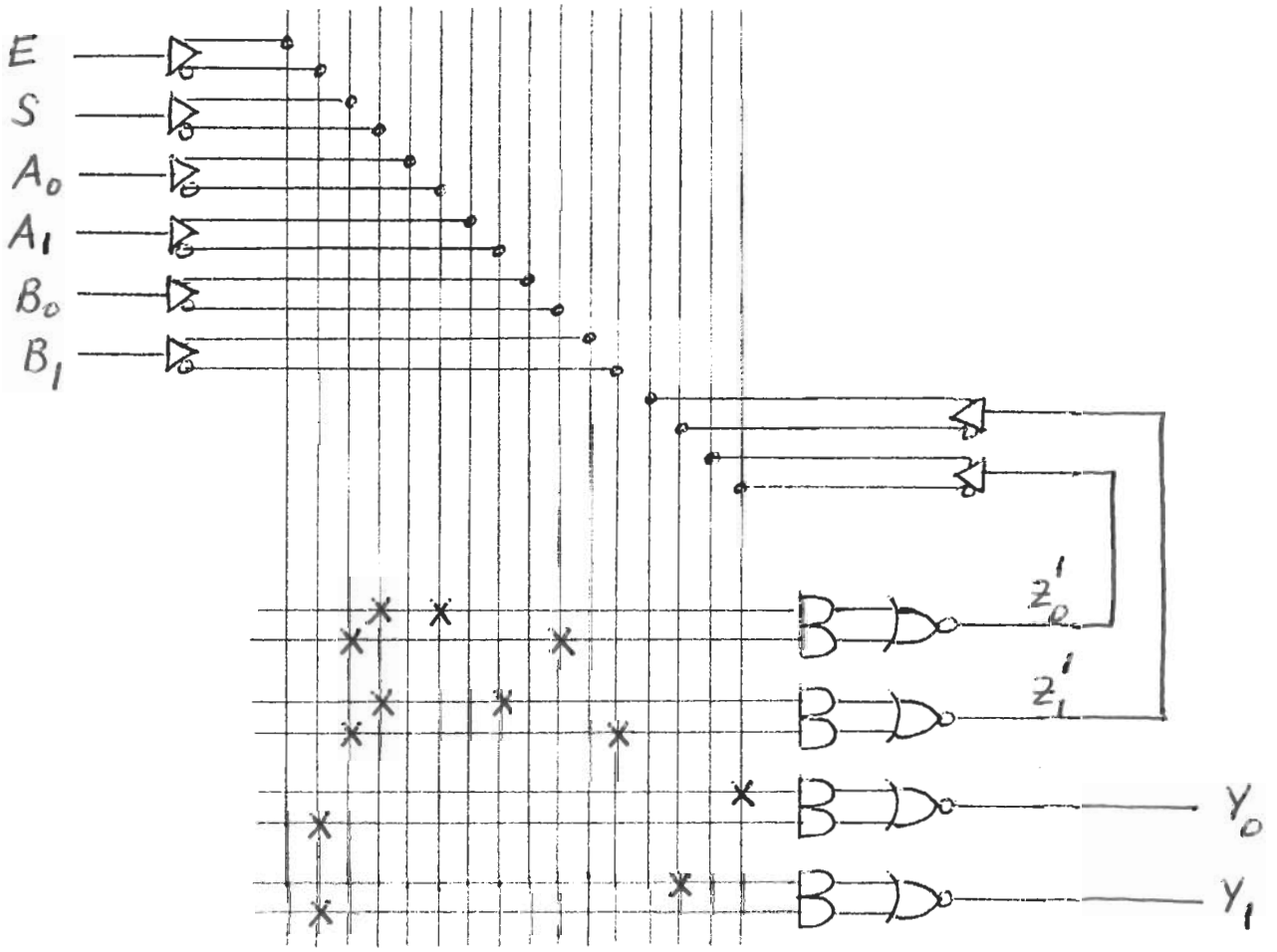
   c. [15 pts] Design the control unit of your ASM using a minimum number of D-type flip-flops and combinational circuit elements. **Note:** You do not need to provide the inputs to the registers in your control unit, just the states and the inputs for the control unit are enough. You only need to design the hardware for the control unit, NOT for the other parts of your machine.

1. Below is the block diagram and the function table of a 2-to-1 2-bit generic multiplexer with enable input.



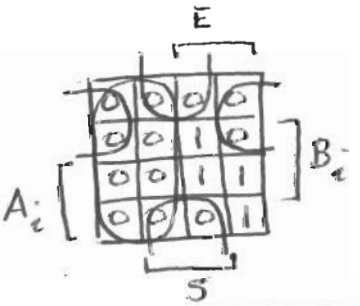
E	S	Y1	Yϕ
0	X	0	0
1	0	A1	Aϕ
1	1	B1	Bϕ

Implement this multiplexer using the following PAL without using any additional gates. Note that PAL outputs are inverted.



SOLUTION :

$$Y_i = E_i(S'A_i + S.B_i)$$



$$Y_i' = E' + S'A_i' + SB_i'$$

$$Z_i$$

2. Implement a 1-bit full adder using a generic 4-to-1 2-bit multiplexer using no more than one additional gate.

SOLUTION

MUX :

S_1	S_0	F_0	F_1
0	0	A_0	A_1
0	1	B_0	B_1
1	0	C_0	C_1
1	1	D_0	D_1

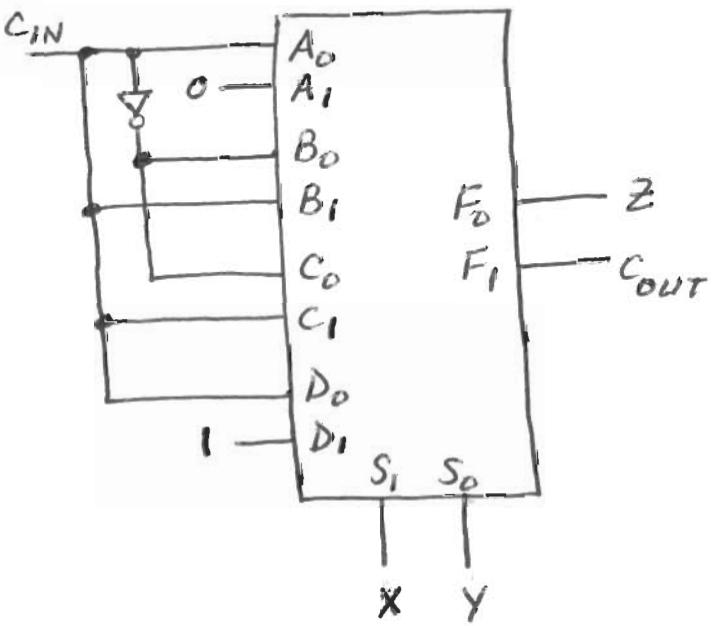
FULL ADDER :

X	Y	C_{IN}	Z	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

Let $XY = S_1 S_0$, $Z = F_0$, $C_{OUT} = F_1$.

FULL ADDER :

S_1	S_0	F_0	F_1
0	0	C_{IN}	0
0	1	C_{IN}'	C_{IN}
1	0	C_{IN}'	C_{IN}
1	1	C_{IN}	1



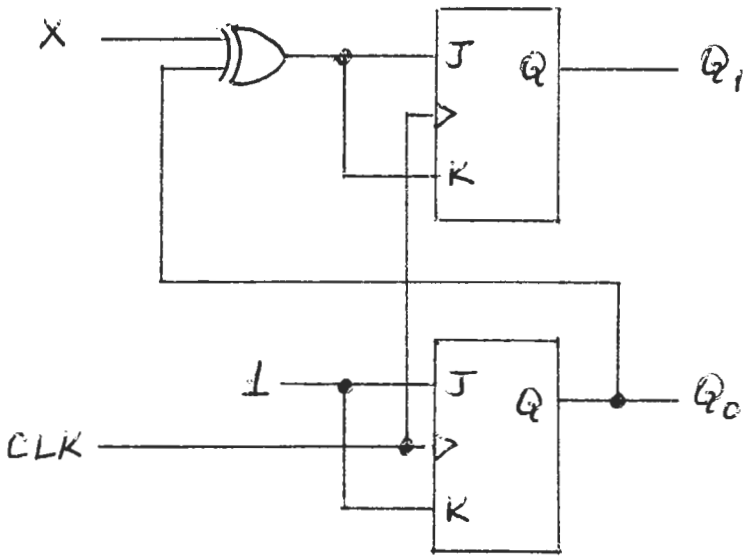
3. Consider the following synchronous machine.

a) Draw the state diagram with the following state assignment

Q_1	Q_0	State
0	0	A
0	1	B
1	0	C
1	1	D

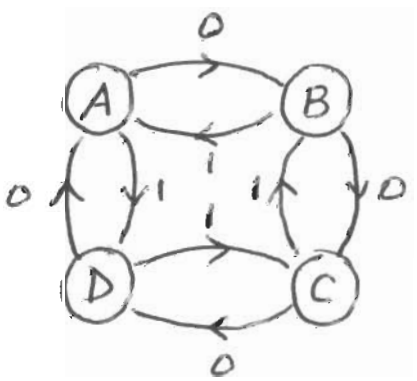
b) Assuming $Q_1 = Q_0 = 0$ initially, find the state sequence corresponding to the following input sequence.

X :	0	0	0	0	0	1	1	1	0	0	1	0
State :	A	B	C	D	A	D	C	B	C	D	C	D



SOLUTION:

Q_1	Q_0	X	$J_1=K_1$	$J_0=K_0$	Q_1^*	Q_0^*
0	0	0	0	1	0	1
0	0	1	1	1	1	1
0	1	0	1	1	1	0
0	1	1	0	1	0	0
1	0	0	0	1	1	1
1	0	1	1	1	0	1
1	1	0	1	1	0	0
1	1	1	0	1	1	0



4. Design a Mealey machine with one input X and one output Y such that $Y=1$ if the present input is the same as the input two clock periods before, and $Y=0$ otherwise. Use only two D-flip flops. Assume that the initial state is $Q_1 = Q_0 = 0$. A typical input, output sequence is given below.

X :	0	0	0	1	0	0	1	0	1	1	1	0	1	0	0	0
Y :	*	*	1	0	1	0	0	1	1	0	1	0	1	1	0	1

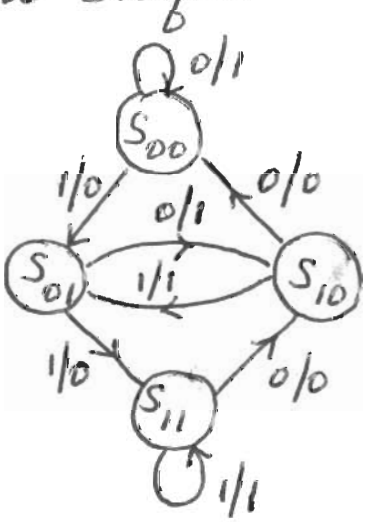
SOLUTION :

States :

S_{00} : Prev. two inputs are 00
S_{01} : " " " " 01
S_{10} : " " " " 10
S_{11} : " " " " 11

} → Use as state codes

State Diagram :



Next State / Output Table :

Q_1	Q_0	X	Q_1^*	Q_0^*	Y
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	1	0	1
0	1	1	1	1	0
1	0	0	0	0	0
1	0	1	0	1	1
1	1	0	1	0	0
1	1	1	1	1	1

$D_0 = Q_0^* = X$

$D_1 = Q_1^* = Q_0$

$Y = (X \oplus Q_1)'$