BILKENT UNIVERSITY
Department of Electrical and Electronics Engineering
EEE102 Introduction to Digital Circuit Design
MidTerm Exam II
SOLUTION
2-5-2005

Q1. (12 points) Implement the function F = AB' + CD' using

a) a generic 8-to-1 multiplexer using extra inverters only, if necessary.

b) a generic 4-to-1 multiplexer using extra inverters and one extra AND or OR gate only, if necessary.

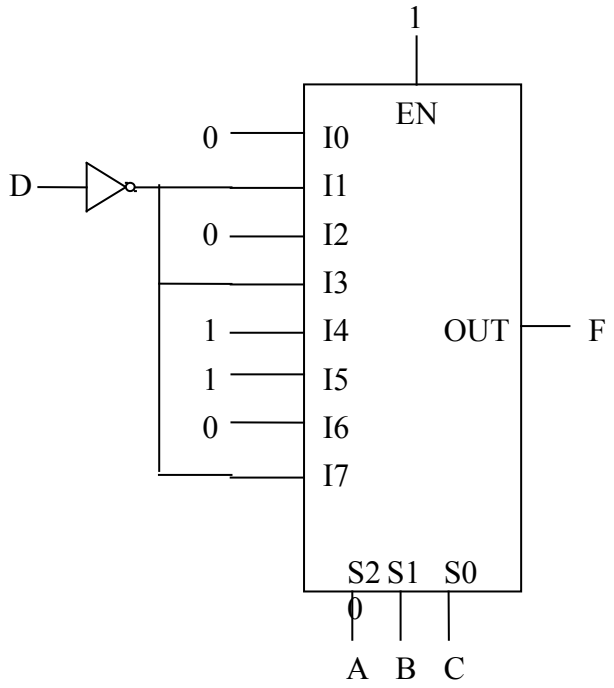
c) two 2-to-4 generic decoders, using one axtra OR gate, if necessary.

In your solutions label all terminals and all internal and external signals.

Solution:

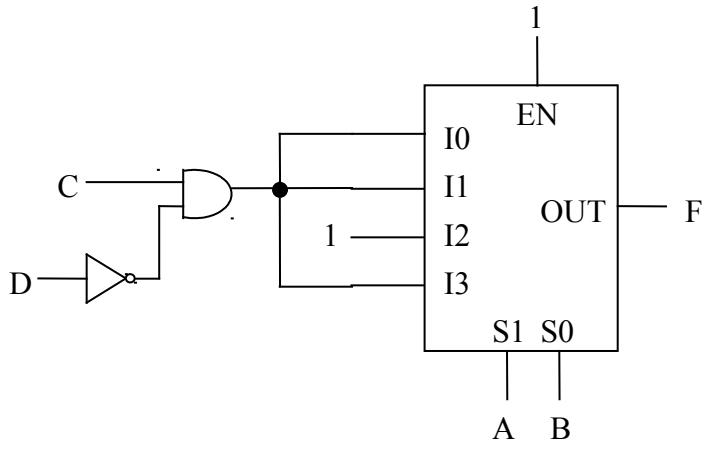
a)

There are other solutions

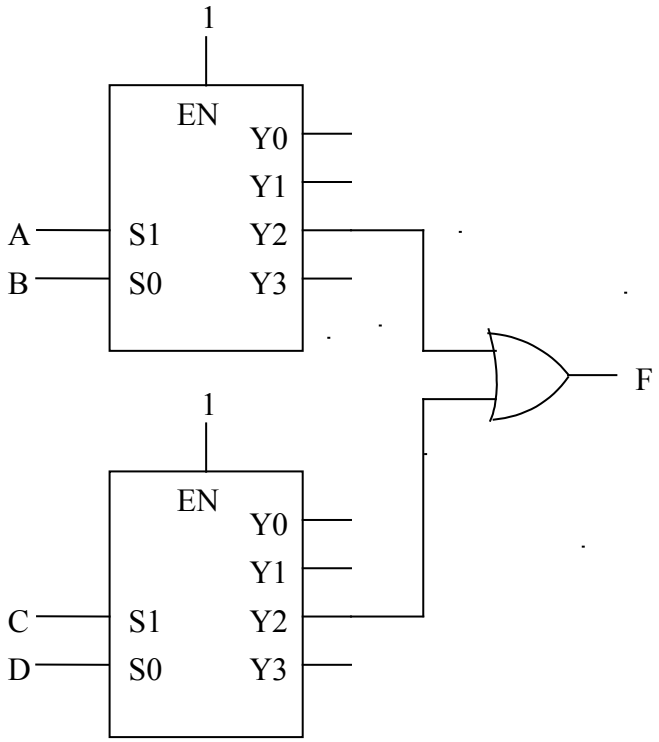


b)

There are other solutions

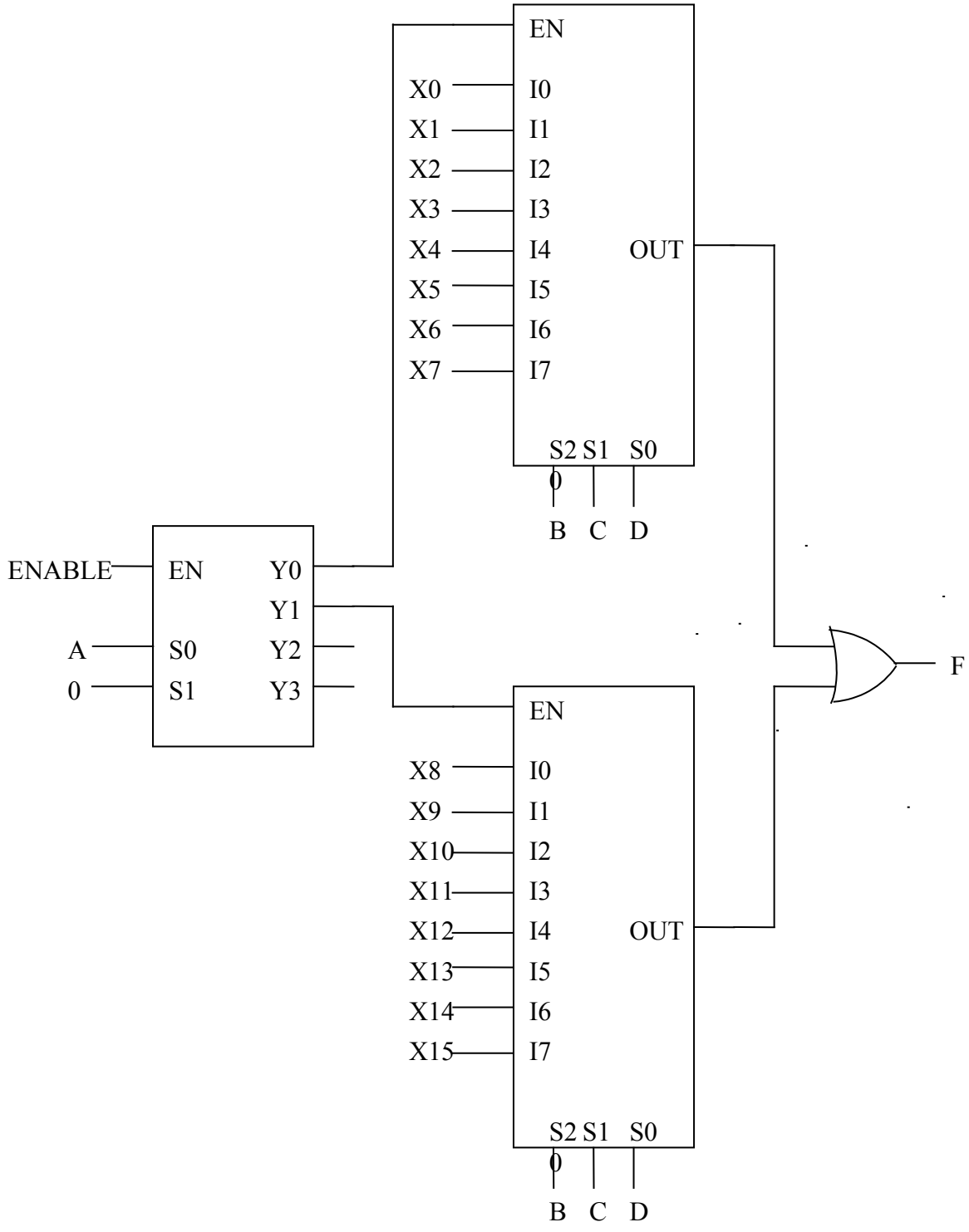


c)



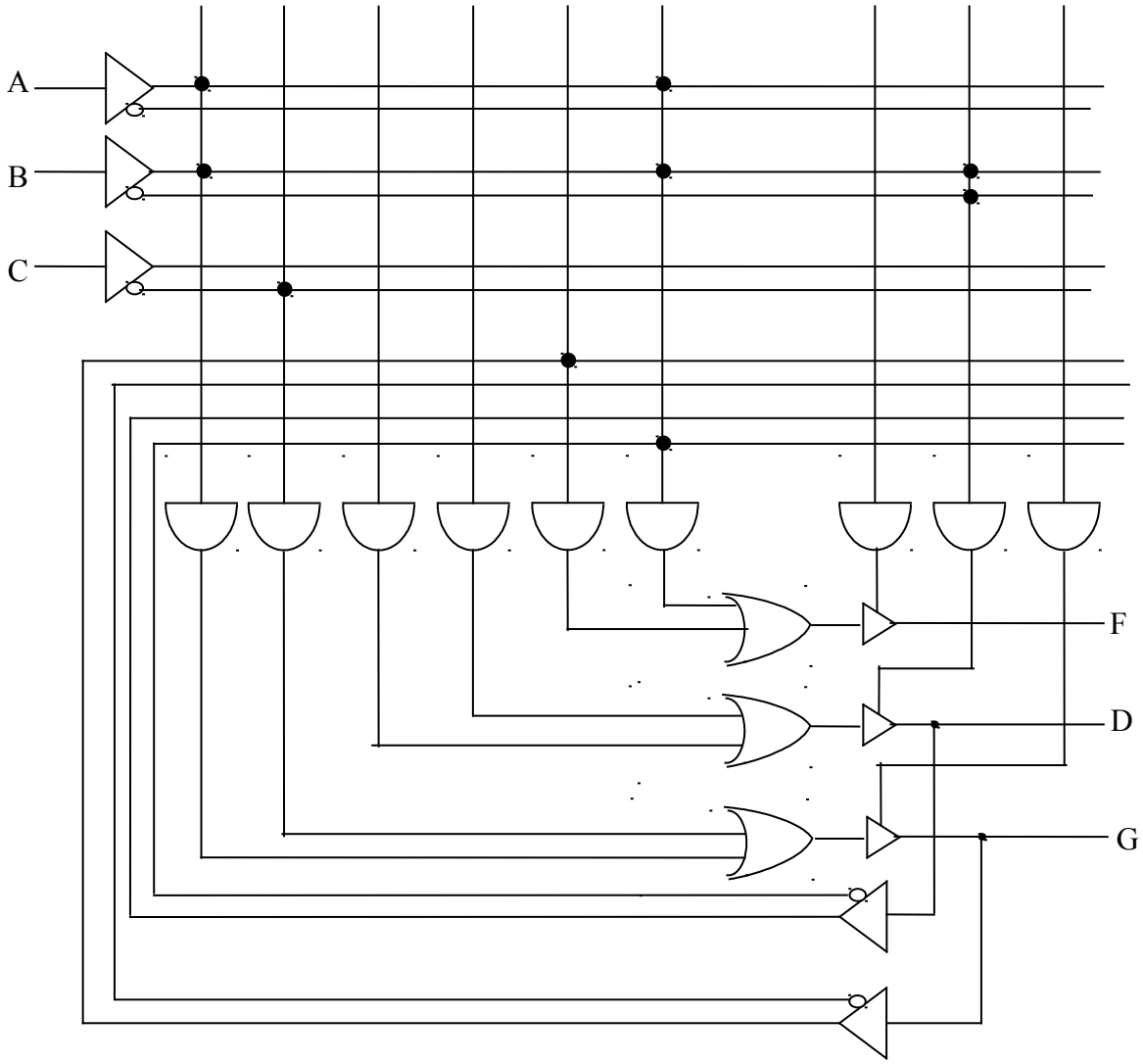
Q2. (13 points) Design and draw a 16-to-1 multiplexer using two 8-to-1 multiplexers and one 2-to-4 decoder. The 8-to-1 multiplexers and the 2-to-4 decoder are generic in the sense that they have one enable each and all of their signals are active high. The 16-to-1 multiplexer that you will design will also be generic with one enable and active high signals. You can use minimum number of additional simple gates.

Solution:



Q3. (15 points) Implement the function F = AB+C'D+ABD' using the PAL shown below (do not change the function). Show the fuses which are to be left intact, and also show all the outside connections if any.

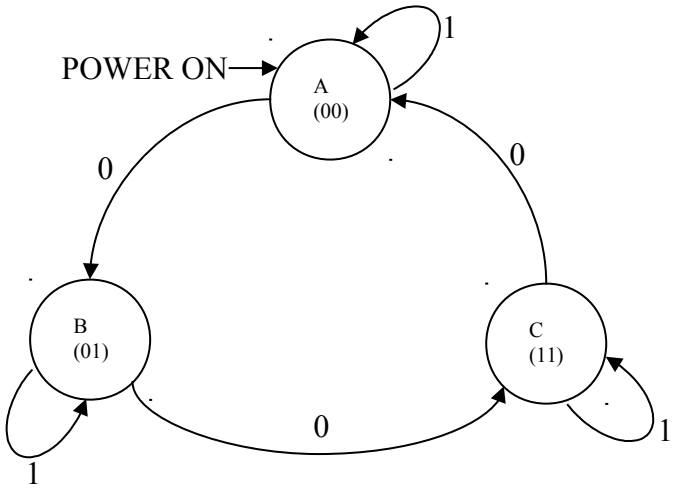
Solution:



G = AB+C'D => F = G + ABD'

Q4. (15 points) A synchronous clocked FSM has 3 states called A, B and C. The machine starts with state A at Power ON. This FSM has an input called HOLD. IF HOLD = '1' at a clock tick then the machine does not change state. If HOLD = '0' however, with the positive edge of the clock signal it goes to B if it is in A, goes to C if it is in B, and goes to B if it is in C. It has 2 outputs Y_1 and Y_0. Y_1Y_0 is "00" if the FSM is in state A, is "01" if in state B and is "10" if in state C. Design and draw this FSM using D ffs. Show all your design steps including state/output diagram, state encoding, next state table, output table, excitation table and minimization of the combinatorial circuits. Draw your circuit. Use RESET IC for initialization. (By proper labeling you need not draw long lines, but draw the gates.)

Solution:



3 states => 2 flip flops. Call their output Q1 and Q0.

State Encoding

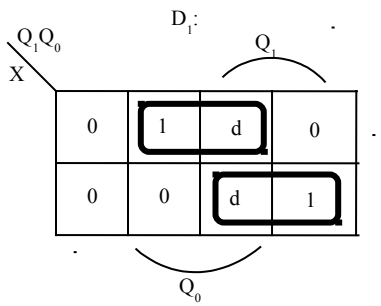
	Q1	Q0
A	0	0
B	0	1
C	1	0

Next State Table. X is for HOLD

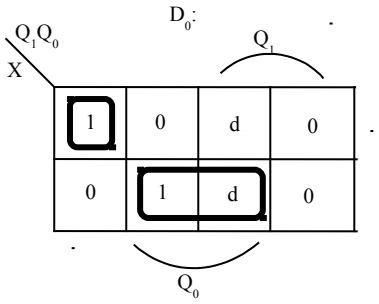
	Q1	Q0	X	Q1*	Q0*
A	0	0	0	0	1
A	0	0	1	0	0
B	0	1	0	1	0
B	0	1	1	0	1
C	1	0	0	0	0
C	1	0	1	1	0
	1	1	0	d	d
	1	1	1	d	d

Excitation table is the same with Q1* replaced by D1 and Q0* by D0.

Minimized Next State Logic



$D_1 = Q_0X' + Q_1X$



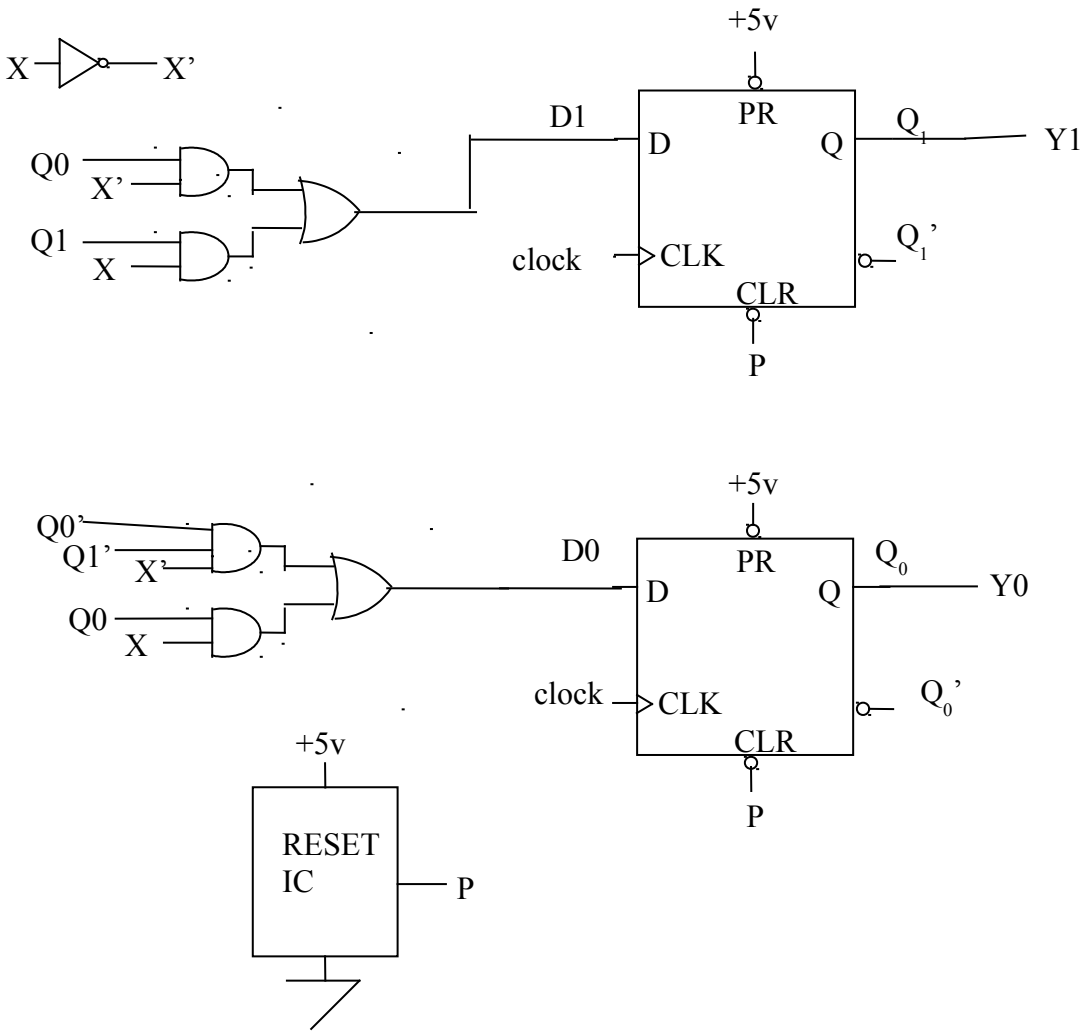
$D_0 = Q_1'Q_0'X' + Q_0X$

Output Table

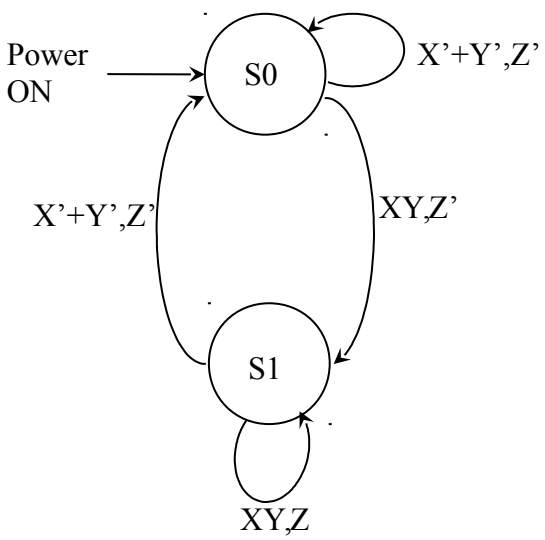
	Q1	Q0		Y1	Y0
A	0	0		0	0
B	0	1		0	1
C	1	0		1	0
	1	1		d	d

Therefore
Y1 = Q1 and Y0 = Q0

Circuit including initialization



Q5. (15 points) A Mealy FSM has the following state/output diagram. This FSM has two inputs X and Y, and a single output Z. On each arrow there are two expressions separated by a comma. The first expression is about the inputs and the second expression is about the output. These expressions are used to express the values of X, Y and Z in compact form. For example XY means X=1 and Y=1. In other words these are the values of X and Y which make the expression XY TRUE, that is, 1. Similarly Z' means the output is 0 because if Z=0 then the expression Z' is TRUE, that is 1.



Design and draw this FSM using JK ffs. Show all your design steps including state encoding, next state table, output table, excitation table and minimization of the combinatorial circuits. Draw your circuit. Use RESET IC for initialization. (By proper labeling you need not draw long lines, but draw the gates.)

Solution:

2 states . 1 ff. Call its output Q.

State Encoding

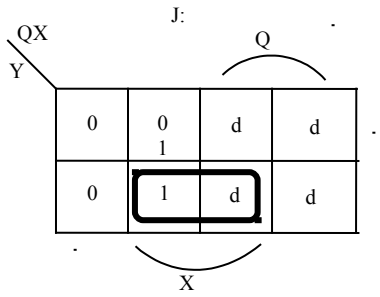
	Q
S0	0
S1	1

Next State and Excitation Table together.

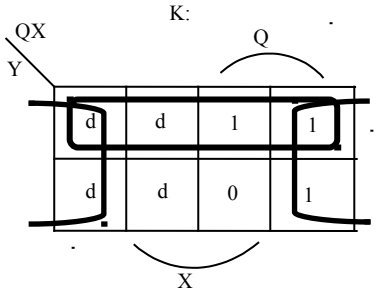
	Q	X	Y	Q*	J	K
S0	0	0	0	**0**	**0**	**d**
S0	0	0	1	**0**	**0**	**d**
S0	0	1	0	**0**	**0**	**d**
S0	0	1	1	**1**	**1**	**d**
S1	1	0	0	**0**	**d**	**1**
S1	1	0	1	**0**	**d**	**1**
S1	1	1	0	**0**	**d**	**1**
S1	1	1	1	**1**	**d**	**0**

Minimized Next state Logic

^



J = XY



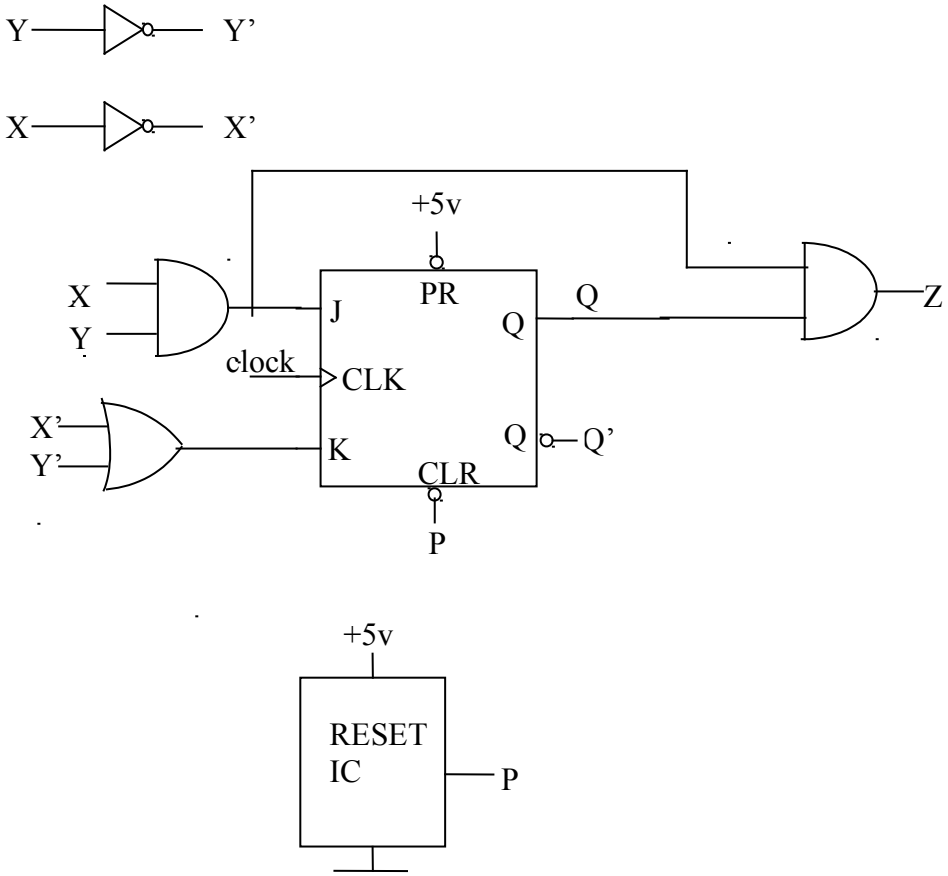
K = X'+Y'

Output Table

	Q	X	Y	Z
S0	0	0	0	**0**
S0	0	0	1	**0**
S0	0	1	0	**0**
S0	0	1	1	**0**
S1	1	0	0	**0**
S1	1	0	1	**0**
S1	1	1	0	**0**
S1	1	1	1	**1**

Z = QXY

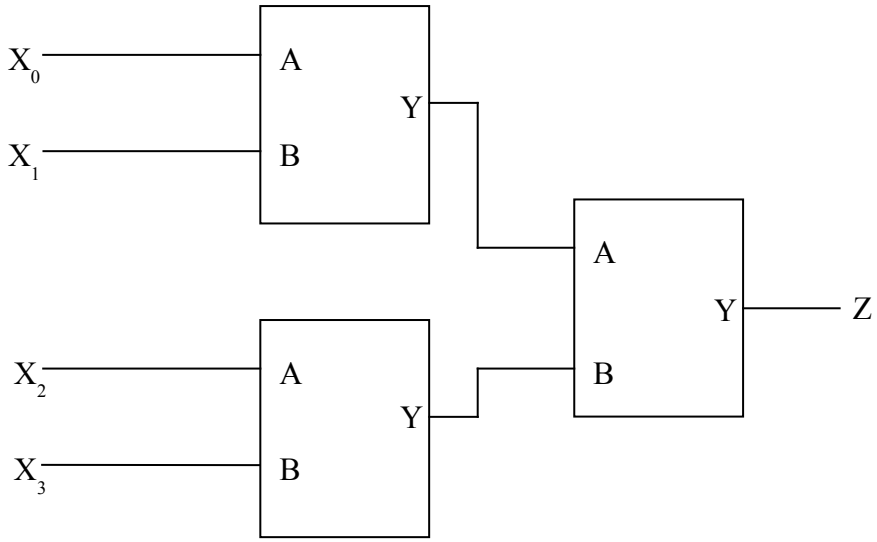
Circuit with initialization:



Q6. (15 points) The VHDL code for an entity called BRICK is already written and included in the library. The entity definition for BRICK is as follows.

```
entity BRICK is
        port(A,B:in std_logic;
             Y:out std_logic );
end BRICK;
```

Write VHDL code for the circuit below. Some VHDL language templates are given in the Appendix.



Solution:

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

entity top is
   Port ( X : in std_logic_vector(3 downto 0);
          Z : out std_logic);
end top;

architecture Structural of top is
 component BRICK
 Port ( A,B: in std_logic;
        Y : out std_logic);
 end component;
 signal U,V:std_logic;
begin
  label1:BRICK port map(X(0),X(1),U);
  label2:BRICK port map(X(2),X(3),V);
  label3:BRICK port map(U,V,Z);
end Structural;
```

Q7. (15 points) Write VHDL code for a 2-to-4 decoder which has two enables, one active low and one active high, and active low outputs. Some VHDL language templates are given in the Appendix.

Solution:

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

entity dec2to4 is
   Port ( EN : in std_logic;
          EN_L : in std_logic;
          S : in std_logic_vector(1 downto 0);
          Y : out std_logic_vector(3 downto 0));
end dec2to4;

architecture Behavioral of dec2to4 is
signal TEMP: std_logic_vector(3 downto 0);
begin
 with S select
          TEMP<="1110" when "00",
                        "1101" when "01",
                        "1011" when "10",
                        "0111" when "11",
                        "1111" when others;
          Y<=TEMP when EN='1' and EN_L='0' else "1111";

end Behavioral;
```

Another solution:

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

entity dec2to4_another is
   Port ( EN : in std_logic;
          EN_L : in std_logic;
          S : in std_logic_vector(1 downto 0);
          Y : out std_logic_vector(3 downto 0));
end dec2to4_another;

architecture Behavioral of dec2to4_another is
begin
  process(EN,EN_L,S)
  begin
            if EN='1' and EN_L = '0' then
               case S is
               when "00" => Y<="1110";
               when "01" => Y<="1101";
```

```
            when "10" => Y<="1011";
            when "11" => Y<="0111";
            when others => Y<="1111";
            end case;
        else Y<= "1111";
        end if;
  end process;
end Behavioral;
```

Yet another solution:

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

entity dec2to4_yet_another is
    Port ( EN : in std_logic;
           EN_L : in std_logic;
           S : in std_logic_vector(1 downto 0);
           Y : out std_logic_vector(3 downto 0));
end dec2to4_yet_another;

architecture Behavioral of dec2to4_yet_another is
begin
  process(EN,EN_L,S)
  begin
       if EN='1' and EN_L = '0' then
            if S="00" then Y<="1110";
            elsif S="01" then Y<="1101";
            elsif S="10" then Y<="1011";
            elsif S="11" then Y<="0111";
            else Y<="1111";
            end if;
       else Y<="1111";
       end if;
  end process;
end Behavioral;
```

Appendix: Some VHDL language templates are given below:

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*);

---

for-generate LOOP

label: for *identifier* in *range* generate
concurrent-statement
end generate;

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*;

CONCURRENT SIGNAL-ASSIGNMENT STATEMENT

signal_name <= expression;
or
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*;

---

process STATEMENT

process(*signal_name*, *signal_name, ..., signal_name*)
  type_declarations
  variable declarations
  constant declarations
begin
  sequential-statement
  …
  sequential-statement
end process;

---

if STATEMENTS

if *boolean_expression* then *sequential_statement*
end if;

if *boolean_expression* then *sequential_statement*
else *sequential_statement*
end if;

if *boolean_expression* then *sequential_statement*
elsif *boolean_expression* then *sequential_statement*
…
elsif *boolean_expression* then *sequential_statement*
else *sequential_statement*
end if;

---

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;
