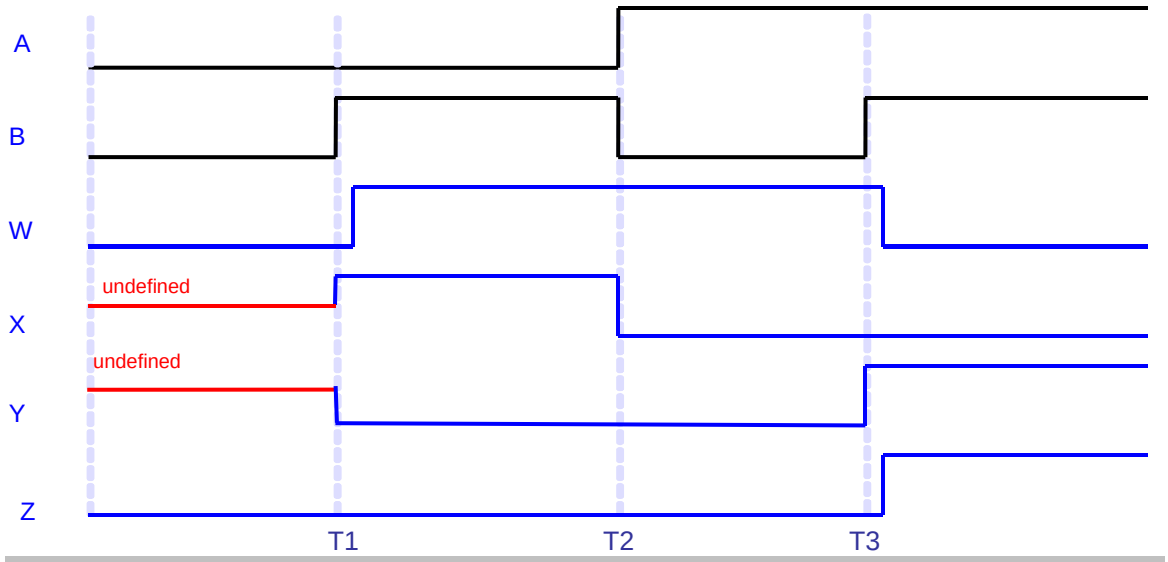
Tutorial PART 2 (SOLUTIONS)

NOTE THAT THERE ARE OTHER POSSIBLE SOLUTIONS TOO…

SS1

Process only triggered/executed when there is an event on B (ie at T1,T2,T3). Note during the execution of process, signals do not change values while the variables can change. At T1,T2,T3: X changes based on "old value" of W while Z follows "new value" of Y. Also note the propagation delays for the output signals!



SS2 (possible solutions)

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

-- library call needed when using std logic functions
-- entity declaration

entity ss2 is
   port ( A,B,C,D : in  STD_LOGIC;
          X, Z : out  STD_LOGIC);
end ss2;

architecture ss2arch of ss2 is
begin

  X <= A xor B xor C;

  Z <= (not (A) and B) or (not(B) and not(C)  and D) or (not(B and D));

end ss2arch;
```

Question 1

Behavioral modeling

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
-- Z is 1 if A is a non-prime number
entity combcct is
   port( A : in std_logic_vector (3 downto 0); Z: out std_logic );
end combcct;

architecture beh of combcct is
begin
          process ( A )
            begin
               case (A) is
                     when "0010" => Z <='0';  -- 2 is a prime number
                     when "0011" => Z <='0';
                     when "0101" => Z <='0';
                     when "0111" => Z <='0';
                     when "1011" => Z <='0';
                     when "1101" => Z <='0';
                     when others =>  Z <='1';
                end case;
          end process;
end beh;
-- any other solutions..?
```

Question 2

```
entity customized_mux is
   port( A, B, C, D,E: in  std_logic_vector(3 downto 0);
         S             : in  std_logic_vector(2 downto 0);
         T             : out std_logic_vector(3 downto 0)  );
end customized_mux;

architecture prog1 of customized_mux is  -- architecture 1
begin                                    - - in dataflow style of modeling
     with S select
               T <=   A        when "000",
                      B        when "001",
                      A        when "010",
                      C        when "011",
                      A        when "100",
                      D        when "101",
                      A        when "110",
                      E        when others;
end prog1;

architecture prog2 of customized_mux is  -- architecture 2
begin                                   – in behavioral style of modeling
process ( S, A,B,C,D,E )
```

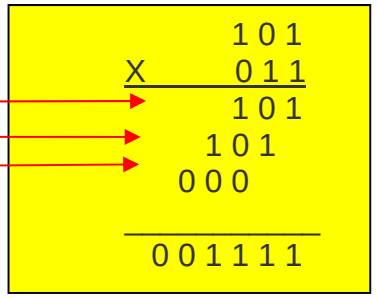
```
variable tmp: std_logic_vector(3 downto 0);
begin
   case S is
                when “000” => tmp:=A;
                when “001” => tmp:=B;
                when “010” => tmp:=A;
                when “011” => tmp:=C;
                when “100” => tmp:=A;
                when “101” => tmp:=D;
                when “110” => tmp:=A;
                when others => tmp:=E;
     end case;
  T <= tmp;
  end process;
end prog2;
```

Question 3

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

entity Dev_74138 is
port (E1, E2, E3: in std_logic;
      A : in std_logic_vector (2 downto 0);
      Z : out std_logic_vector (0 to 7));  -- note that  the order is Z(0), Z(1), Z(3)…. Z(7).
end Dev_74138;
architecture BEH of Dev_74138 is
 begin
  process (A, E1, E2, E3)    -- how would this work if A is not in the list? Would it still be a combinational cct?
   variable tmp : std_logic_vector (0 to 7);
   begin
            if     E1 = '1' then Z <= "11111111";
            elsif E2 = '1' then Z <= "11111111";
            elsif E3 = '0' then Z <= "11111111";
            else
              case A is
                        when “000” => tmp:= "01111111";
                        when “001” => tmp:= "10111111";
                        when “010” => tmp:= "11011111";
                        when “011” => tmp:= "11101111";
                        when “100” => tmp:= "11110111";
                        when “101” => tmp:= "11111011";
                        when “110” => tmp:= "11111101";
                        when “111” => tmp:= "11111110";
                       when others=>  tmp:= "11111111";
               end case;
               Z <= tmp;
            end if;
    end process;
end BEH;
```

Question 4

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity mul_beh is                                          -- behavioral style of modeling
   Port ( x : in  STD_LOGIC_VECTOR (2 downto 0);
          y : in  STD_LOGIC_VECTOR (2 downto 0);
          z : out  STD_LOGIC_VECTOR (5 downto 0));
end mul_beh;

architecture arch_mul_beh of mul_beh is
begin
          process( x , y )
          variable w1,w2,w3, temp : std_logic_vector(5 downto 0);
          begin
                    w1 := "000" & x(2) & x(1) & x(0);
                    w2 := "00" & x(2) & x(1) & x(0) & '0';   --w1*2
                    w3 := '0' & x(2) & x(1) & x(0) & "00";   --w1*4
                    case y is
                              when "000" => temp:= "000000";
                              when "001" => temp:= w1;
                              when "010" => temp:= w2;
                              when "011" => temp:= w2+w1;
                              when "100" => temp:= w3;
                              when "101" => temp:= w3+w1;
                              when "110" => temp:= w3+w2;
                              when others => temp:= w3+w2+w1;
                            end case;
                    Z<=temp;
   end process;  -- using if  statements to do the adding is another way to solve this!
end arch_mul_beh;
```



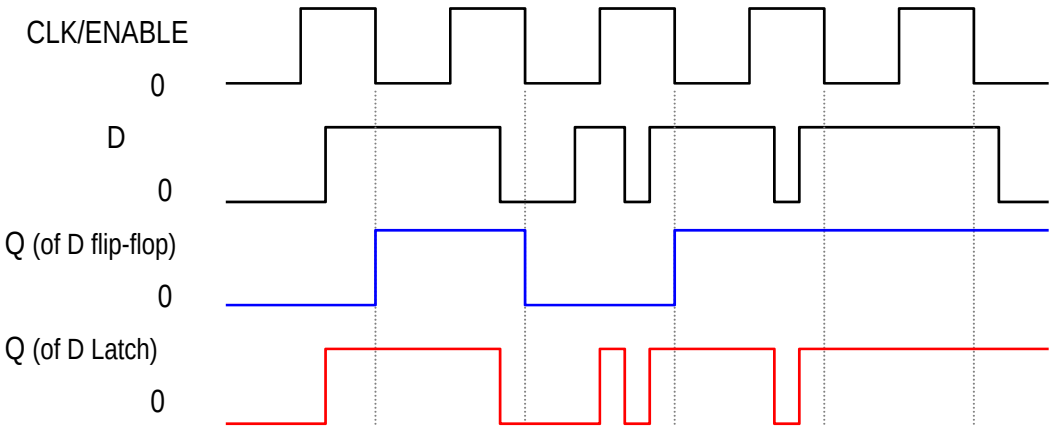
Question 5

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

entity Dev_Encrpt is
port (D_in : in std_logic_vector (4 downto 0);     D_out : out std_logic_vector (2 downto 0)  );
end Dev_Encrpt;

architecture BEH of Dev_Encrpt is
 begin
  process (D_in)    -- work out another solution in dataflow style!!
   begin
          if     D_in(3) = '0' and D_in(1) = '0' then D_out <= (D_in(4), D_in(2),D_in(0) );
          elsif D_in(3) = '0' and D_in(1) = '1' then D_out <= (D_in(0), D_in(4),D_in(2) );
          elsif D_in(3) = '1' and D_in(1) = '0' then D_out <= (D_in(2), D_in(0),D_in(4) );
          else                                  D_out <= ( not(D_in(4)) , not(D_in(2)), not( D_in(0)) );
          end if;
    end process;
end BEH;
```

SS3

```
library IEEE; -- VHDL program which realizes the circuit of Question 5 of Tutorial 5 (Part 1).
use IEEE.STD_LOGIC_1164.ALL;

entity t5 is
   Port ( B : in std_logic_vector ( 3 downto 0); N : out  std_logic_vector (3 downto 0));
end t5;

architecture Behavioral of t5 is
begin
   process(B)
    variable temp: std_logic_vector (3 downto 0);
      begin
       case B is
            when "0000" => temp:="0011";
            when "0001" => temp:="0111";
            when "0010" => temp:="1000";
            when "0011" => temp:="0010";
            when "0100" => temp:="0110";
            when "0101" => temp:="1001";
            when "0110" => temp:="0000";
            when "0111" => temp:="1111";
            when "1000" => temp:="0101";
            when "1001" => temp:="0001";
            when "1010" => temp:="1010";
            when "1011" => temp:="0100";
            when "1100" => temp:="1011";
            when "1101" => temp:="1101";
             when "1110" => temp:="1100";
             when others => temp:="1110";
      end case;
     N <= temp;
   end process;
end Behavioral;
```

Question SS4

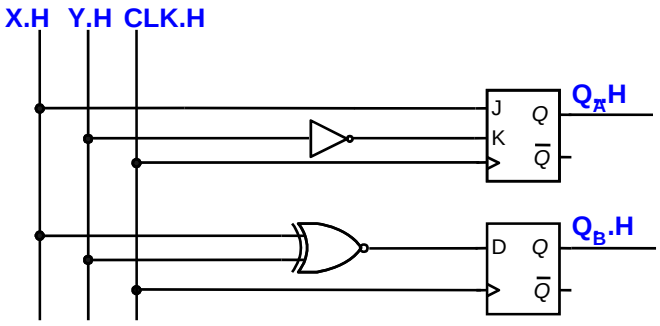


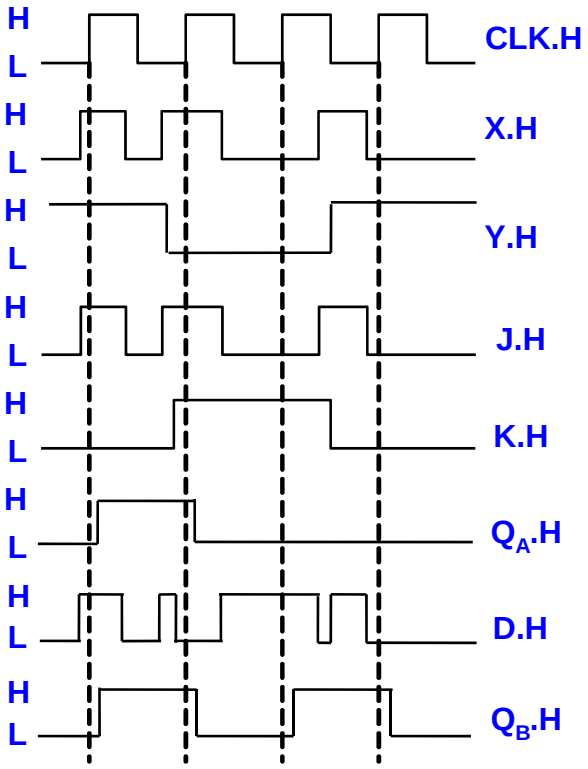
Assumption: The D input satisfies the set-up and hold times of the Latch and flip-flop. Also, the propagation delay of the devices are much less than the CLK period.

Question 6

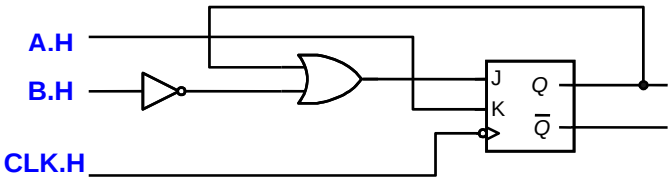
Connect D to J and $\overline{D}$ to K.

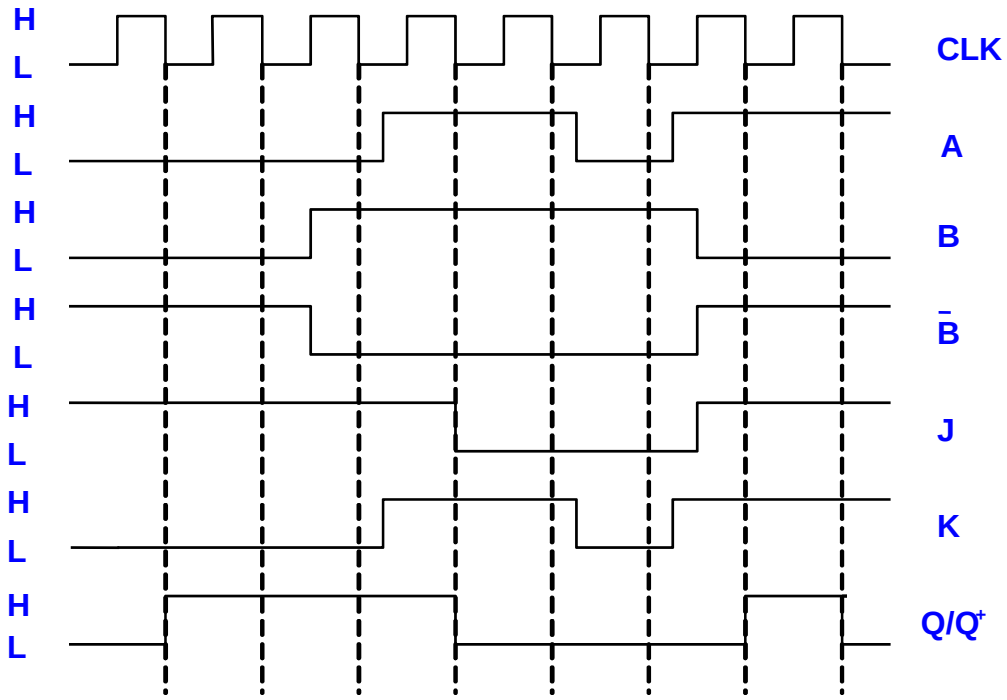
Question SS5





Question 7





A	B	Q	Q^+
0	0	0	**1**
0	0	1	**1**
0	1	0	**0**
0	1	1	**1**
1	0	0	**1**
1	0	1	**0**
1	1	0	**0**
1	1	1	**0**

Characteristic Table

A	B	Q^+
0	0	**1**
0	1	**Q**
1	0	$\overline{Q}$
1	1	**0**

Condensed Characteristic Table

Q	Q^+	A	B
0	0	X	1
0	1	X	0
1	0	1	X
1	1	0	X

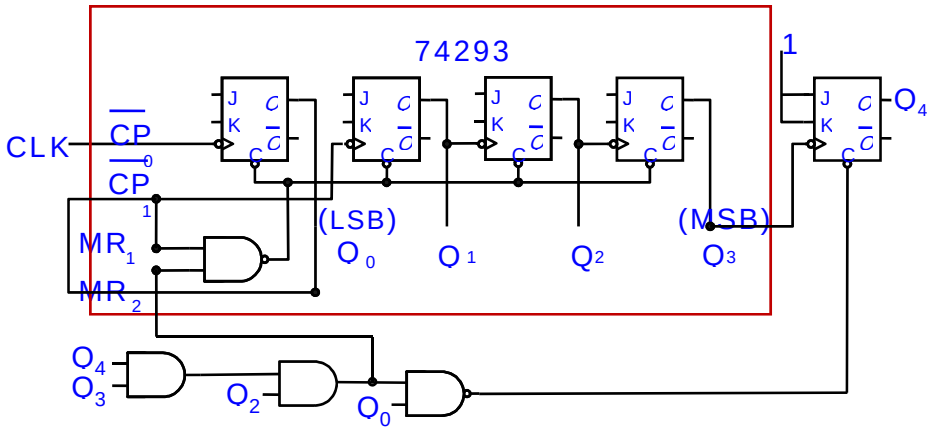
Excitation Table

Question 8

We need an extra JK flip-lop to implement a 5-bit counter. Such a counter would count from 0 to 31 and so to implement a mod29 counter, it needs to be cleared when count reaches 29:

$$29_{10} = \begin{matrix} Q_4 & Q_3 & Q_2 & Q_1 & Q_0 \\ (1 & 1 & 1 & 0 & 1)_2 \end{matrix}$$

Since "11111" will not be reached for a mod-29 counter, the condition to be checked for is "111X1" i.e., we only need to detect $Q_4 = Q_3 = Q_2 = Q_0 = 1$ to clear the counter. *How about for a Mod-27 counter?*



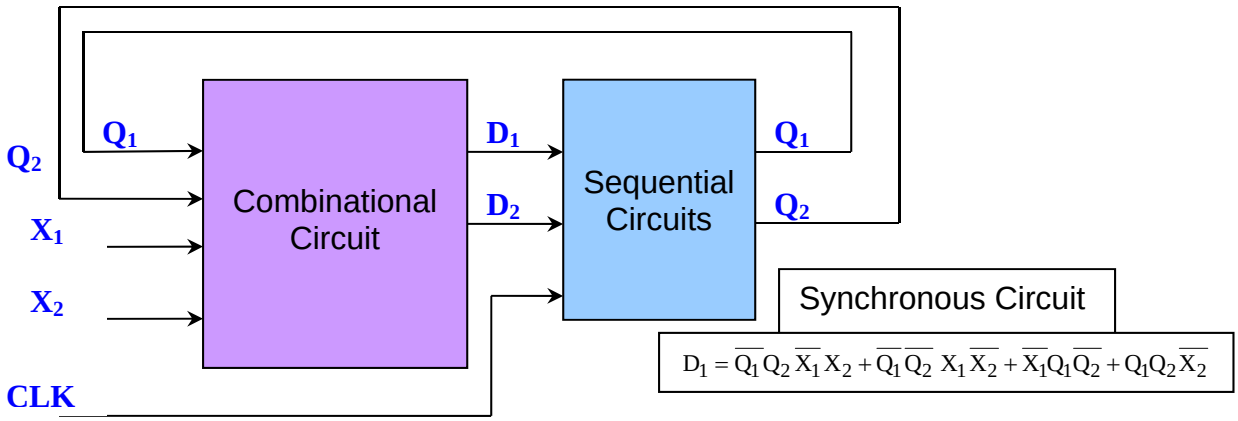
Question 9 (next page)

Question 10

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

entity count01_beh is
   Port (x, clk, rst : in std_logic;       y : out std_logic);
end count01_beh;
architecture count01_beh_arch of count01_beh is
begin
        process( clk , rst)                          -- assume that x synchronously changes with clock
          variable zeros_count: std_logic_vector (1 downto 0):= "00";   -- zeros_count
          variable ones_count: std_logic_vector (1 downto 0):= "00";    -- ones_count
        begin
                if (rst = '1') then y <= '0'; zeros_count := "00"; ones_count := "00";
                elsif (clk'event and clk = '1') then
                        y <= '0';  --assigning a default value; else y will be 'U' until it becomes 1 or is reset.
                        if     (x = '0') and (zeros_count<"11") then zeros_count := zeros_count +'1';
                        elsif (x = '1') and (ones_count<"11")  then ones_count := ones_count + '1';
                        end if;
                        if (zeros_count = "11" and ones_count = "11") then y <= '1'; end if;
                end if;
        end process;
end count01_beh_arch;
```

Question 9

4 mode 2-bit counter



$$D_1 = \overline{Q_1}Q_2\overline{X_1}X_2 + \overline{Q_1}\overline{Q_2}\,X_1\overline{X_2} + \overline{X_1}Q_1\overline{Q_2} + Q_1Q_2\overline{X_2}$$

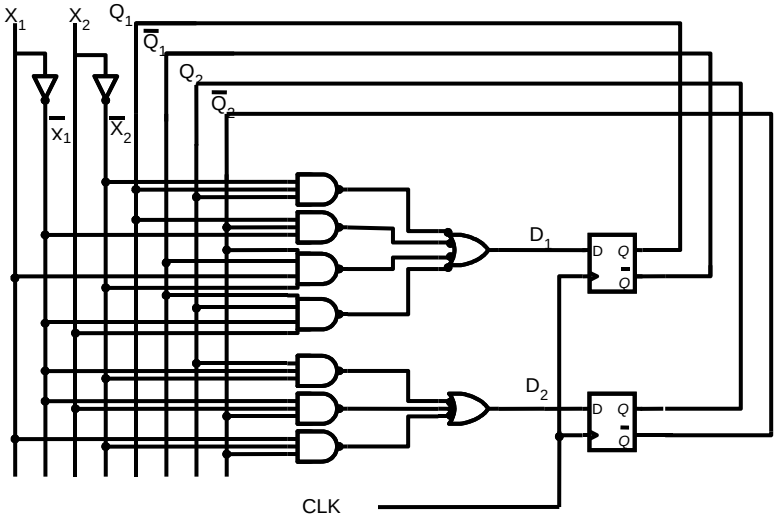
The next state table is:

X_1	X_2	Q_1	Q_2	Q_1^+ / D_1	Q_2^+ / D_2
0	0	0	0	0	0
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	1	1
0	1	0	0	0	1
0	1	0	1	1	0
0	1	1	0	1	1
0	1	1	1	0	0
1	0	0	0	1	1
1	0	0	1	0	0
1	0	1	0	0	1
1	0	1	1	1	0
1	1	0	0	0	0
1	1	0	1	0	0
1	1	1	0	0	0
1	1	1	1	0	0

X_1X_2 \ Q_1Q_2	00	01	11	10
00	0	0	1	1
01	0	1	0	1
11	0	0	0	0
10	1	0	1	0

X_1X_2 \ Q_1Q_2	00	01	11	10
00	0	1	1	0
01	1	0	0	1
11	0	0	0	0
10	1	0	0	1

$$D_2 = Q_2\overline{X_1}\,\overline{X_2} + \overline{X_1}X_2\overline{Q_2} + X_1\overline{X_2}\,\overline{Q_2}$$



Question 11

A	B	Q	Q^+
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

a: Characteristic Table

Q	Q^+	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

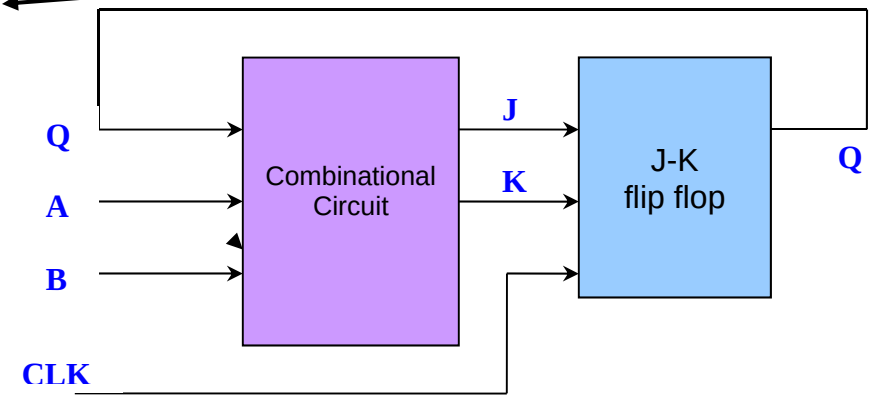
b: Excitation Table

A	B	Q	Q^+	J	K
0	0	0	1	1	X
0	0	1	1	X	0
0	1	0	0	0	X
0	1	1	1	X	0
1	0	0	1	1	X
1	0	1	0	X	1
1	1	0	0	0	X
1	1	1	0	X	1

c: Combined Table

A	B	Q	J	K
0	0	0	1	X
0	0	1	X	0
0	1	0	0	X
0	1	1	X	0
1	0	0	1	X
1	0	1	X	1
1	1	0	0	X
1	1	1	X	1

d: Truth Table for J and K



Functional Block diagram

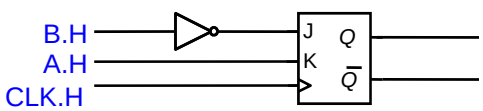
e: Use Karnaugh Maps to find logic expressions of J and K

$J = \overline{B}$

AB \ Q	0	1
00	1	X
01	0	X
11	0	X
10	1	X

$K = A$

AB \ Q	0	1
00	X	0
01	X	0
11	X	1
10	X	1



Attempt Question 6 following a similar procedure!

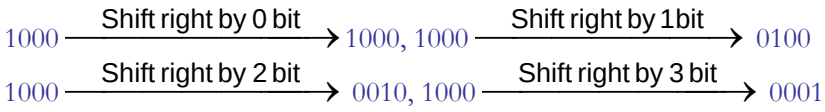
Question 12

Previous value of X	Current value of X	Q+
0	0	Q
0	1	0
1	0	1
1	1	$\overline{Q}$

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

entity flipflop_beh is
   Port ( x : in std_logic;       clk : in std_logic;          y : buffer std_logic);
end flipflop_beh;

architecture flipflop_beh_arch of flipflop_beh is
begin

process( clk )
variable xold, xnew, tmp: std_logic := '0';
begin
        if (clk'event and clk = '1') then
                xnew := x; tmp := y;
                if (xnew = '0' and xold = '1') then tmp := '1';
                        elsif (xnew = '1' and xold = '0') then tmp := '0';
                        elsif (xnew = '1' and xold = '1') then tmp := not tmp;
                end if;
                xold := xnew;
        y <= tmp;
        end if;
end process;
end flipflop_beh_arch;
```

Question 13

A four-bit barrel shifter that can shift to the right by 0, 1, 2 or 3 positions.

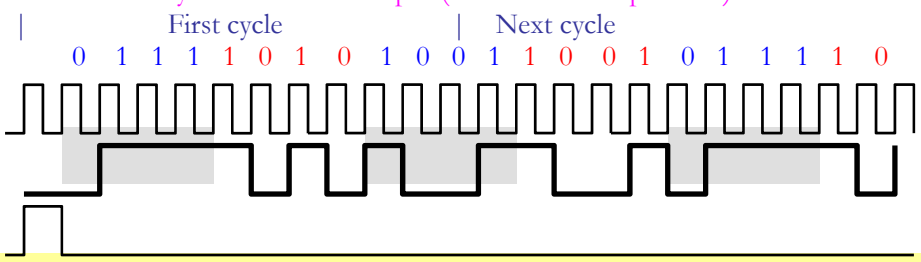



```
-- This particular barrel shifter is implemented as a combinational circuit
-- with a 4 bit input "data" and a 4 bit output "data_sft".
-- The "shift" input is a two bit vector that specifies whether
-- to shift the input by 0, 1, 2 or 3 positions to the right.

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity barrlshft_beh is
    Port ( data : in std_logic_vector(3 downto 0);
           shift : in std_logic_vector(1 downto 0);
           data_sft : out std_logic_vector(3 downto 0));
end barrlshft_beh;
```

```
architecture barrlshft_beh_arch of barrlshft_beh is
begin
process(data, shift)
begin
       case shift is
              when "00" =>   data_sft <= data;
              when "01" =>   data_sft <= '0' & data(3) & data(2) & data(1);
              when "10" =>   data_sft <= "00" & data(3) & data(2);
              when others => data_sft <= "000" & data(3);
       end case;
end process;
end barrlshft_beh_arch;
```

Question 14

The timing diagram of the pulse generator is as shown below. They are defined as follows:
Waveform 1: Clock pulse
Waveform 2: Serial output for "1001 1001 0101 1110"
Waveform 3: Synchronous load input (Indicates load operation)




```
library IEEE;                          -- library declaration
use IEEE.STD_LOGIC_1164.ALL;           -- package declarations
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity serial_sft is
    Port ( clk,rst,start_load : in std_logic;
           data : in std_logic_vector(15 downto 0);  -- 16 bit input data
           shift_out : out std_logic);        -- the bit being shifted out
end serial_sft;
--start_load loads the new data when asserted. Also wakes the system up from idle state
architecture serial_sft_arch of serial_sft is
       type states is (idle, serial_sft);    --declaring the various states
       signal state: states;
       signal count: std_logic_vector(3 downto 0);

begin
 process (clk,rst)            --asynchronous reset
 variable tmp: std_logic_vector(15 downto 0);
   begin
   if rst='1' then           -- enter idle state if system is reset
      state <= idle;  shift_out <= '0'; tmp := "0000000000000000";  count <= "0000";
   elsif clk'event and clk = '1' then
       case state is
       when idle =>
              if start_load = '1' then  -- parallel load the data & reset counter
                     state <= serial_sft; tmp:=data;
                     shift_out <= '0';count <= "0000";
              end if;

       when serial_sft =>
              if start_load='1' then
                     tmp:=data; shift_out <= '0'; count <= "0000";
                     else shift_out <= tmp(conv_integer(count)); count := count+1;
              end if;    -- conv_integer is defined in library and required to
                         -- convert a vector to integer before it can be used as an index
       end case;
   end if;
 end process;
end serial_sft_arch;
```

Question 15

The sequence is: …0,2,6,7,5,0,2,6,7,5…. Thus, the next state table is as follows:

CURRENT STATE			NEXT STATE		
$Q2^N$	$Q1^N$	$Q0^N$	$Q2^{N+1}$	$Q1^{N+1}$	$Q0^{N+1}$
0	0	0	0	1	0
0	1	0	1	1	0
1	0	1	0	0	0
1	1	0	1	1	1
1	1	1	1	0	1
all other entries			X	X	X

However, as the PAL16R8 outputs are inverted,

$Q2^N$	$Q1^N$	$Q0^N$	$\overline{Q2}^{N+1}$	$\overline{Q1}^{N+1}$	$\overline{Q0}^{N+1}$
0	0	0	1	0	1
0	1	0	0	0	1
1	1	1	0	1	0
1	0	1	1	1	1
1	1	0	0	0	0
all other entries :			X	X	X

The D FF inputs are:

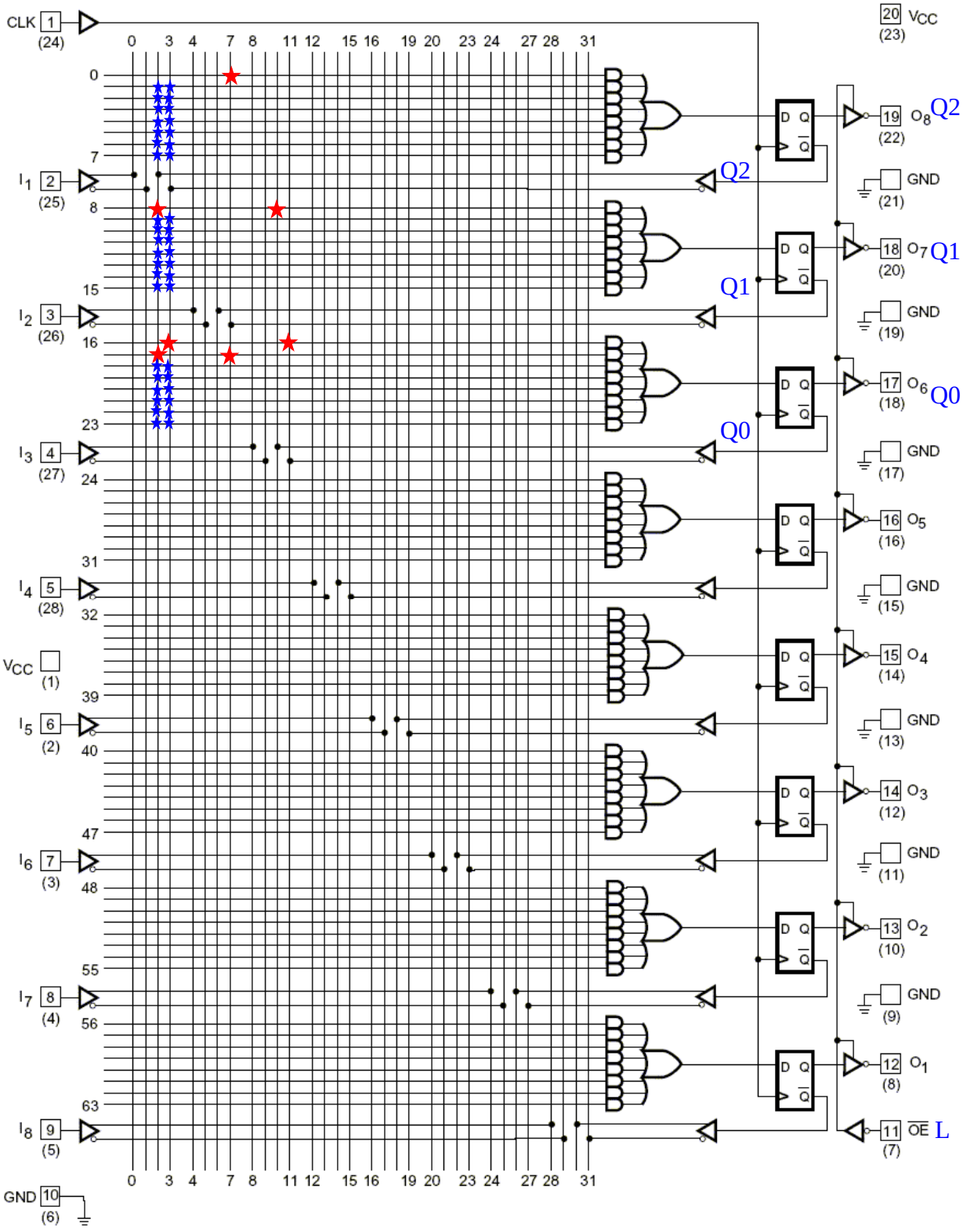
D-FF #2 = $\overline{Q1}$

D-FF #1 = $\underline{Q0}\ \underline{Q2}$

D-FF #0 = $Q2\ Q0 + Q2\ \overline{Q1}$

Please note that only a rough simplification is done in the above solution, and the D-FF input expressions *can* be simplified further. However, we *need not* simplify if the resources (number of product terms etc) available on the PLD is sufficient to realize the circuit.
OR-connections of the PAL are *not* programmable. To ensure that the output of the AND gates which are not in use remains de-asserted (low), we need the connections (★) as shown in the figure.

16R8


CLK 1 (24)
I1 2 (25)
I2 3 (26)
I3 4 (27)
I4 5 (28)
VCC (1)
I5 6 (2)
I6 7 (3)
I7 8 (4)
I8 9 (5)
GND 10 (6)
20 VCC (23)
19 O8 (22)
GND (21)
18 O7 (20)
GND (19)
17 O6 (18)
GND (17)
16 O5 (16)
GND (15)
15 O4 (14)
GND (13)
14 O3 (12)
GND (11)
13 O2 (10)
GND (9)
12 O1 (8)
11 OE (7)
Q2
Q1
Q0
L
0 3 4 7 8 11 12 15 16 19 20 23 24 27 28 31
0
7
8
15
16
23
24
31
32
39
40
47
48
55
56
63

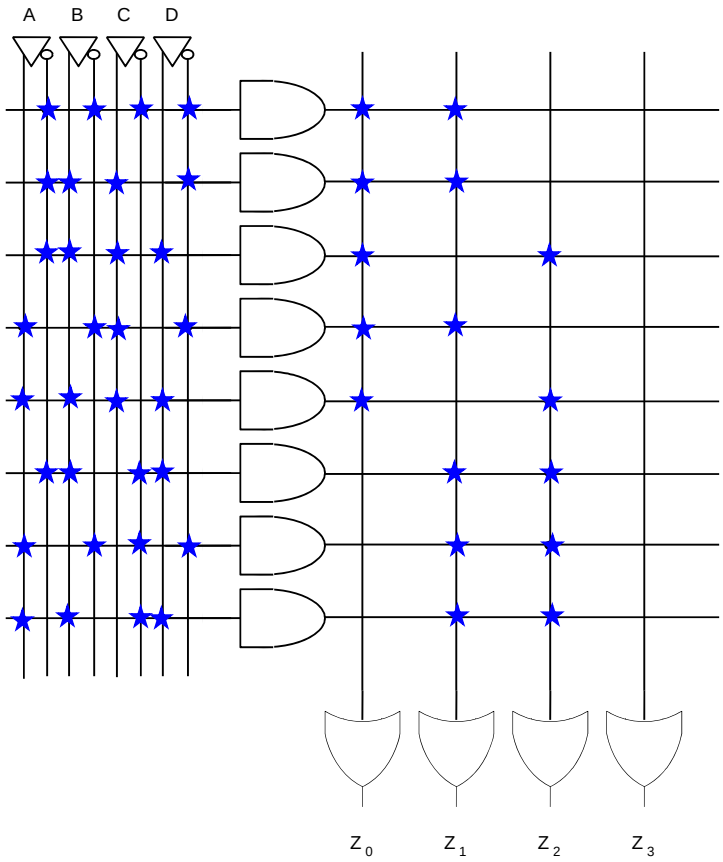
Question 16

A	B	C	D	Z_0	Z_1	Z_2
0	0	0	0	1	1	0
0	0	0	1	0	0	0
0	0	1	0	0	0	0
0	0	1	1	0	0	0
0	1	0	0	0	0	0
0	1	0	1	0	1	1
0	1	1	0	1	1	0
0	1	1	1	1	0	1
1	0	0	0	0	1	1
1	0	0	1	0	0	0
1	0	1	0	1	1	0
1	0	1	1	0	0	0
1	1	0	0	0	0	0
1	1	0	1	0	1	1
1	1	1	0	0	0	0
1	1	1	1	1	0	1

$$Z_0 = \overline{A}\,\overline{B}\overline{C}\overline{D} + \overline{A}BC\overline{D} + \overline{A}BCD + A\overline{B}C\overline{D} + ABCD$$

$$Z_1 = \overline{A}\,\overline{B}\overline{C}\overline{D} + \overline{A}B\overline{C}D + \overline{A}BC\overline{D} + A\overline{B}\overline{C}\overline{D} + A\overline{B}C\overline{D} + AB\overline{C}D$$

$$Z_2 = \overline{A}\,B\overline{C}D + \overline{A}BCD + A\overline{B}\overline{C}\overline{D} + AB\overline{C}D + ABCD$$



Question 17

3K module needs to have 12 address lines A_{11}-A_0 and devices need to be enabled as follows:

A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**2K RAM**
1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	**Module**
1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	**1K RAM**
1	0	0	0	1	0	1	1	1	1	1	1	1	1	1	1	**Module**
1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	**Unused**
1	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	**memory**

{unshaded bits applied to address inputs of the RAM devices}

Capacity of memory unit = *Number of memory locations* + *Number of bits per location*
= $(2^{11} + 2^{10})$ x 8 {i.e., 3K x 8 bits }
= 24 576

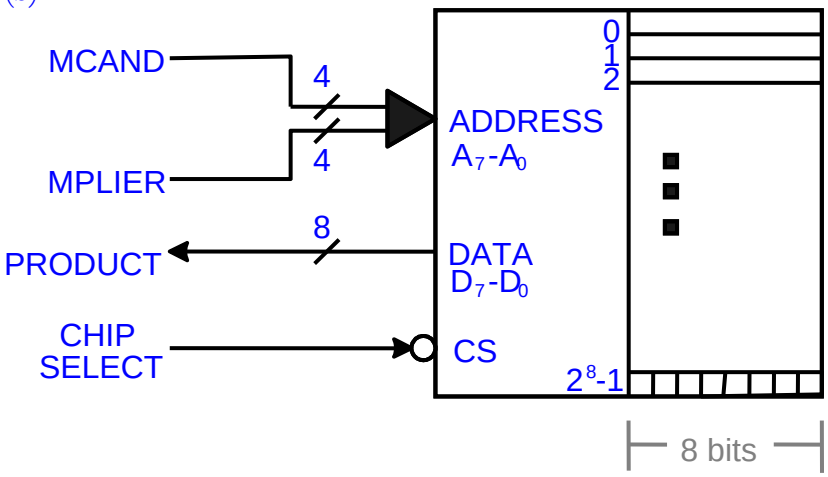
Memory range in use = Initial memory location ↔ Final memory Location
= (8000H + 0000H) ↔ (8000H + 0BFFH)
= 8000H ↔ 8BFFH

Unused memory range = (8000H + 0C00H) ↔ (8000H + 0FFFH)
= 8C00H ↔ 8FFFH
(assuming that the memory map is till 8FFFH)

Question 18

(a). ROM capacity = 2^8 x 8

(b).



(c). The ROM contains 2^8 rows of 8 bit data. Each row is indexed by an address represented by MCAND and MPLIER and contains an 8 bit data of the corresponding PRODUCT.

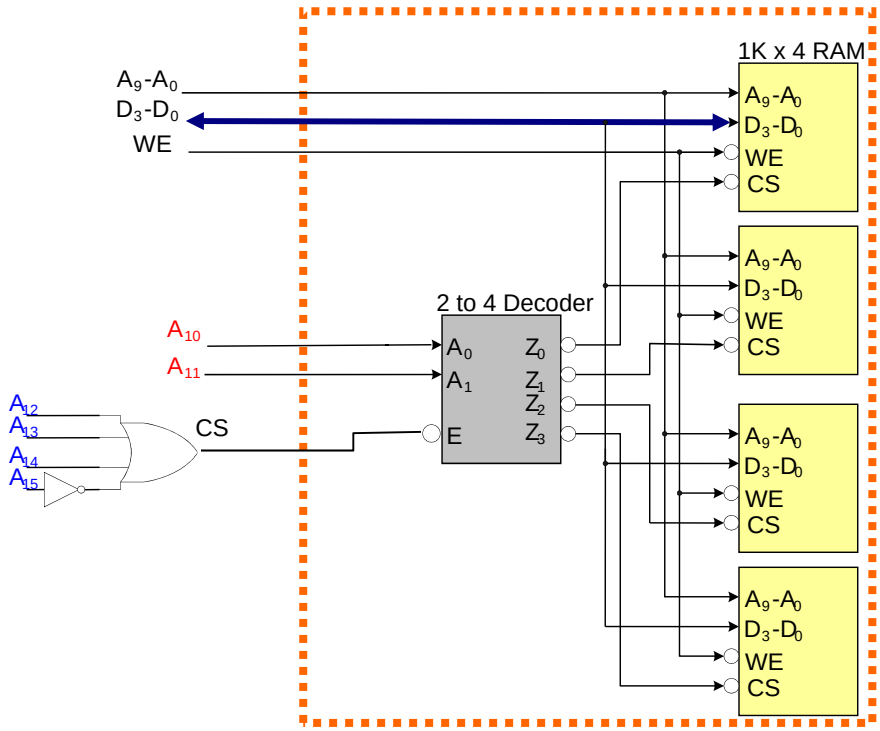
Question 19

A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1K RAM
1	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	Device1
1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1K RAM
1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	Device2
1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1K RAM
1	0	0	0	1	0	1	1	1	1	1	1	1	1	1	1	Device3
1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1K RAM
1	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	Device 4

From the above table, when A12 to A15 is "1000" the RAM module (i.e., one of the RAM devices) is enabled:

- the decoder is enabled
- A10 to A11 determines which decoder output (and thus RAM device) is active
- A0 to A9 applied to the address inputs of each RAM selects the particular location within the **selected** RAM device.

4K x 4 RAM module (dotted box) with memory address starting from 8000H



Is it possible to achieve exhaustive decoding by giving A_1 -A_0 *as input to the 2-4 decoder instead of* A_{11} –A_{10}*?*