

**Module -1** Principles of combinational logic-1**Hrs: 10**

Definition of combinational logic, Canonical forms, Generation of switching equations from truth tables, Karnaugh maps-3, 4 and 5 variables, Incompletely specified functions (Don't Care terms), Simplifying Max term equations.

**Recommended readings:**

1. John M Yarbrough, "Digital Logic Applications and Design",  
Thomson Learning, 2001.

Unit-3.1, 3.2, 3.3, 3.4

notes4free.in

### combinational logic

Also known as "combinatorial logic," it refers to a digital logic function made of primitive logic gates (AND, OR, NOT, etc.) in which all outputs of the function are directly related to the current combination of values on its inputs. Any changes to the signals being applied to the inputs will immediately propagate through the gates until their effects appear at the outputs. Contrast with sequential logic.

### sequential logic

A digital logic function made of primitive logic gates (AND, OR, NOT, etc.) in which the output values depend not only on the values currently being presented to its inputs, but also on previous input values. The output depends on a "sequence" of input values. Contrast with combinational logic.

### Canonical Forms

There are two standard or *canonical* ways of expressing boolean functions:

#### 1. *Sum-of-products (SOP)*.

E.g.

$$X = A + \overline{B}C + \overline{A}BC$$

#### 2. *Product-of-sums (POS)*

E.g.

$$X = (A + D)(C + \overline{D})(\overline{B} + C)$$

These representations are useful for

- direct implementation, and
- starting logic function minimization.

We will focus on SOP.

Consider

$$f(A, B, C) = A + \overline{B}C + \overline{A}BC$$

where

- *product terms*  $A, \overline{B}C, \overline{A}BC$
- *minterms*  $\overline{A}BC$

A minterm is any ANDed term containing all of the variables (perhaps complemented).

Let's look at the truth table which corresponds to this function:

	A	B	C	$f(A,B,C)$
$m_0$	0	0	0	0
$m_1$	0	0	1	1
$m_2$	0	1	0	0
$m_3$	0	1	1	1
$m_4$	1	0	0	1
$m_5$	1	0	1	1
$m_6$	1	1	0	1
$m_7$	1	1	1	1

(Check this!)

Each row of the truth table corresponds to one of the  $2^n = 8$  possible minterms in  $n=3$  variables.

$$m_i = m_i(A, B, C), \quad i = 0, \dots, 2^3 - 1$$

E.g.

$$m_3 = \bar{A}BC = 001$$

Actually, the truth table specifies the function as a *sum of minterms*:

$$\begin{aligned} f(A, B, C) &= m_1 + m_3 + m_4 + m_5 + m_6 + m_7 \\ &= \sum m(1, 3, 4, 5, 6, 7) \end{aligned}$$

This is called the *canonical SOP representation* of the function  $f$ .

The *minterm code* for  $n=3$  is as follows:

$m_0$	0	0	0	$\bar{A}$	$\bar{B}$	$\bar{C}$
-------	---	---	---	-----------	-----------	-----------

$m_1$	0	0	1	$\bar{A}$	$\bar{B}$	$C$
$m_2$	0	1	0	$\bar{A}$	$B$	$\bar{C}$
$m_3$	0	1	1	$\bar{A}$	$B$	$C$
$m_4$	1	0	0	$A$	$\bar{B}$	$\bar{C}$
$m_5$	1	0	1	$A$	$\bar{B}$	$C$
$m_6$	1	1	0	$A$	$B$	$\bar{C}$
$m_7$	1	1	1	$A$	$B$	$C$

Complemented variables correspond to 0 and un complemented variables correspond to 1.

$$f(A, B, C) = A + \bar{B}C + \bar{A}BC$$

The function can be put into canonical SOP form algebraically as follows:

$$\bar{B}C = (\bar{A} + A)\bar{B}C = \bar{A}\bar{B}C + A\bar{B}C = m_1 + m_5$$

$$A = (B + \bar{B})A = \dots = m_4 + m_5 + m_6 + m_7$$

$$f(A, B, C) = \sum(1, 3, 4, 5, 6, 7)$$

(fill in the missing steps!) and so on combining we get as before.

*Any Boolean function can be expressed in canonical SOP form.*

### **Simplification and Implementation of Boolean Functions**

Boolean functions can be implemented in hardware in a number of ways. For instance, standard discrete TTL or CMOS ICs could be used, in which case it is useful to find the simplest expression for the function being implemented. Or if programmable devices are to be used, then a more direct representation of the function may be useful.

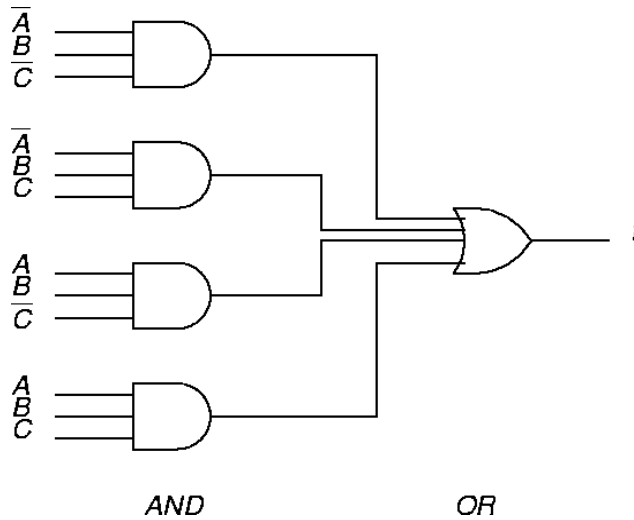
#### **Direct Implementation**

Consider the function

$$f(A, B, C) = \sum m(2, 3, 6, 7)$$

expressed in canonical SOP form. Then assuming all variables and their complements are available we can implement this function with the AND-OR circuit of Figure as shown.

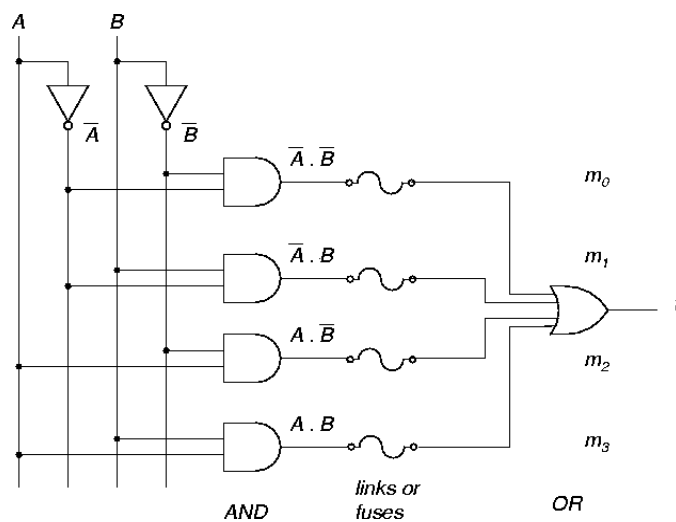
**Figure :** AND/OR implementation.



This implementation is *not minimal* in general (i.e. can realize  $f$  with fewer gates).

This representation is direct and is useful when implementing with **programmable logic devices (PLD)**. To illustrate, consider functions  $f=f(A,B)$  of two variables ( $n=2, 2^n=4$ ). A PLD schematic is shown in Figure.

**Figure :** PLD implementation.



This PLD can program any given function  $f(A,B)$  by breaking appropriate links.

### Karnaugh Maps (K-Maps)

*Karnaugh* or *K-maps* are useful tool for boolean function minimization, and for visualization of the boolean function. In brief,

- K-maps provide a graphical method for minimizing boolean functions via pattern recognition for up to about  $n=6$  variables.
- For larger numbers of variables, there are computer algorithms which can yield near-minimal implementations.
- K-maps are a way of expressing truth tables to make minimization easier. They are constructed from minterm codes.

Consider the boolean function

$$f(A, B) = \sum m(0, 1, 2)$$

The truth table is

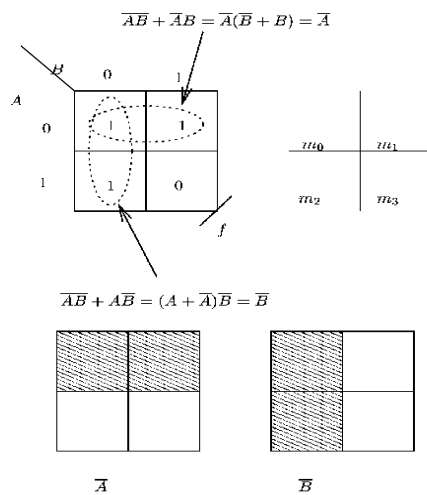
notes4free.in

A	B	f	
0	0	1	$m_0$
0	1	1	$m_1$
1	0	1	$m_2$
1	1	0	$m_3$

The K-map is shown in Figure .The essence of the K-map is the two dimensional representation of  $f$ , which is equivalent to the truth table but more visual.

To minimize  $f$ , we loop out **logical adjacencies**, Figure .

**Figure :** K-map showing looped-out terms and also corresponding minterms.



Therefore

$$f = \overline{A} + \overline{B}$$

This is less complex than  $f$  in canonical SOP form.

**Note.** Looping out logical adjacencies is a graphical alternative to algebraic calculations.

**Unit distance code (Gray code.)** For two bits, the Gray code is:

00 01 11 10

Only one bit changes as you go from left to right. This code **preserves logical adjacencies**.

The **K-map method** is to loop out groups of  $2^n$  logically adjacent minterms. Each looped out group corresponds to a product term in a minimal SOP expression.

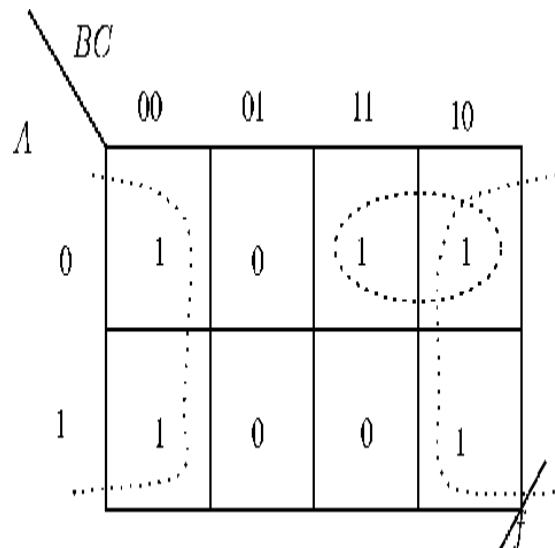
1. Loop out single 1s ( $n=0$ ) which have no logical adjacencies.
  2. Loop out all pairs of 1s ( $n=1$ ) which cannot be included in a larger group.
  3. Loop out all quads of 1s ( $n=2$ ) which cannot be included in a larger group.
- Etc.

$$f(A, B, C) = \sum m(0, 2, 3, 4, 6)$$

**Example.** The K-map is shown in Figure.

$$f(A, B, C) = \sum m(0, 2, 3, 4, 6)$$

**Figure:** K-map for.



Moving left to right or up to down in the K-map changes only one digit in the minterm code. Note the wrap-around at the ends: because of logical adjacency, the top and bottom are joined, and the left and right are joined.

$n=0$ : none

$$m_2 + m_3 = \bar{A}B$$

$n=1$ :

$$m_0 + m_2 + m_4 + m_6 = \bar{C}$$

$n=2$ :

Therefore the minimal SOP representation is

$$f = \bar{A}B + \bar{C}$$

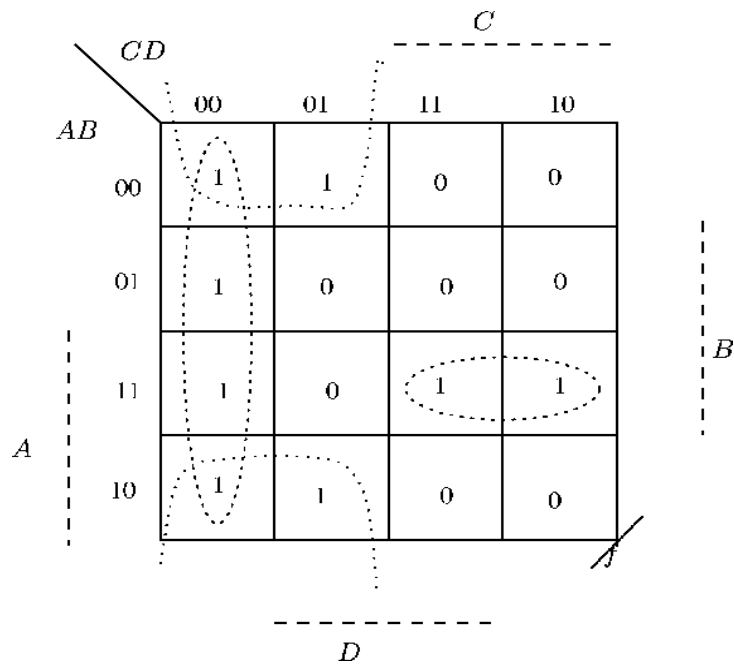
$$f(A, B, C, D) = \sum m(0, 1, 4, 8, 9, 12, 14, 15)$$

**Example.** The K-map is shown in Figure.

$$f(A, B, C, D) = \sum m(0, 1, 4, 8, 9, 12, 14, 15)$$

**Figure:** K-map for.





Therefore the minimal SOP representation is

$$f = ABC + \bar{C}.D + \bar{B}.C$$

notes4free.in

**Don't cares.** In some applications it doesn't matter what the output is for certain input values. These are called *don't cares*.

For instance, in the **Binary Coded Decimal** code, not all input values occur:

0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6

0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15

0, 1, ..., 9

The decimal numbers are those in the range, and a minimum of 4 bits is needed to encode these.

10, 11, ..., 15

The remaining numbers correspond to code values which are not used in BCD.

$\phi$

*We shall use the symbols or X to denote don't cares.*

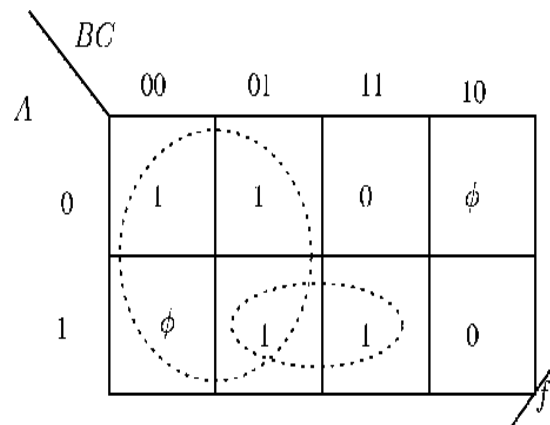
Don't cares can be exploited to help minimize boolean functions.

$$f(A, B, C) = \sum m(0, 1, 5, 7) + \phi(2, 4)$$

**Example.** The K-map is shown in Figure.

$$f(A, B, C) = \sum m(0, 1, 5, 7) + \phi(2, 4)$$

**Figure:** K-map for.



The minimal SOP representation is

$$f = \overline{B} + AC$$

### KARNAUGH MAPS (K-MAP)

A method for graphically determining implicants and implicants of a Boolean function was developed by Veitch and modified by Karnaugh. The method involves a diagrammatic representation of a Boolean algebra. This graphic representation is called map.

It is seen that the truth table can be used to represent complete function of  $n$ -variables. Since each variable can have value of 0 or 1. The truth table has  $2^n$  rows. Each row of the truth table consists of two parts (1) an  $n$ -tuple which corresponds to an assignment to the  $n$ -variables and (2) a functional value.

A Karnaugh map (K-map) is a geometrical configuration of  $2^n$  cells such that each of the  $n$ -tuples corresponds to a row of a truth table uniquely locates a cell on the map. The functional values assigned to the  $n$ -tuples are placed as entries in the cells, i.e. 0 or 1 are placed in the associated cell.

An important feature about the construction of K-map is the arrangement of the cells. Two cells are physically adjacent within the configuration if and only if their respective  $n$ -tuples differ in exactly one element. So that the Boolean law  $x + \overline{x} = 1$  can be applied to adjacent cells. Ex. Two 3-tuples (0,1,1) and (0,1,0) are physically adjacent since these tuples vary by one element.

**One variable** : One variable needs a map of  $2^1 = 2$  cells map as shown below

x f(x)

0 f(0)

1 f(1)

**TWO VARIABLE** : Two variable needs a map of  $2^2 = 4$  cells

x y f(x,y)

0 0 f(0,0)

0 1 f(0,1)

1 0 f(1,0)

1 1 f(1,1)

**THREE VARIABLE** : Three variable needs a map of  $2^3 = 8$  cells. The arrangement of cells are as follows

x y z f(x,y,z)

0 0 0 f(0,0,0)

0 0 1 f(0,0,1)

0 1 0 f(0,1,0)

0 1 1 f(0,1,1)

1 0 0 f(1,0,0)

1 0 1 f(1,0,1)

1 1 0 f(1,1,0)

1 1 1 f(1,1,1)

**FOUR VARIABLE** : Four variable needs a map of  $2^4 = 16$  cells. The arrangement of cells are as follows

w x y z f(w,x,y,z)

0 0 0 0 f(0,0,0,0)

0 0 0 1 f(0,0,0,1)

0 0 1 0 f(0,0,1,0)

0 0 1 1 f(0,0,1,1)

w x y z f(w,x,y,z)

1 0 1 0 f(1,0,1,0)

1 0 1 1 f(1,0,1,1)

1 1 0 0 f(1,1,0,0)

1 1 0 1 f(1,1,0,1)

- 0 1 0 0 f(0,1,0,0)
- 0 1 0 1 f(0,1,0,1)
- 0 1 1 0 f(0,1,1,0)
- 0 1 1 1 f(0,1,1,1)
- 1 0 0 0 f(1,0,0,0)
- 1 0 0 1 f(1,0,0,1)

Four variable K-map.

0000	0001	0011	0010
0100	0101	0111	1010
1100	1101	1111	1110
1000	1001	1011	1010

notes4free.in

Ex. Obtain the minterm canonical formula of the three variable problem given below

$$f(x, y, z) = x y z + x y \bar{z} + x \bar{y} z + x \bar{y} \bar{z}$$

$$f(x, y, z) = \sum m(0, 2, 4, 5)$$

00                      01                      11                      11

1	0	0	1
1	1	0	0

Ex. Express the minterm canonical formula of the four variable K-map given below

00	01	11	10
1	1	0	1
1	1	0	0
0	0	0	0
1	0	0	1

$$f(w,x,y,z) = w x y z + w x y z + w x y z + w x y z + w x y z + w x y z$$

$$f(w,x,y,z) = \sum m(0, 1, 2, 4, 5,$$

Ex. Obtain the max term canonical formula

(POS) of the three variable problem stated above

$$f(x,y,z) = (x + y + z)(x + y + z)(x + y + z)$$

$$f(x,y,z) = \prod M(1,3,6,7)$$

Ex Obtain the max term canonical formula

(POS) of the four variable problem stated above

$$f(w,x,y,z) = (w + x + y + z)(w + x + y + z)(w + x + y + z)$$

$$(w + x + y + z)(w + x + y + z)(w + x + y + z)$$

$$(w + x + y + z)(w + x + y + z)(w + x + y + z)$$

$$f(w,x,y,z) = \prod M(3,6,7,9,11,12,13,14,15)$$

### PRODUCT AND SUM TERM REPRESENTATION OF K-MAP

1. The importance of K-map lies in the fact that it is possible to determine the implicants and implicants of a function from the pattern of 0's and 1's appearing in the map. The cell of a K-map has entry of 1's is referred as 1-cell and that of 0's is referred as 0-cell.

2. The construction of an n-variable map is such that any set of 1-cells or 0-cells which form a  $2^a \times 2^b$  rectangular grouping describing a product or sum term with n-a-b variables, where a and b are non-negative no.

3. The rectangular grouping of these dimensions referred as Sub cubes. The sub cubes must be the power of 2 i.e.  $2^{a+b}$  equals to 1,2,4,8 etc.

4. For three variable and four variable K-map it must be remembered that the edges are also adjacent cells or sub cubes hence they will be grouped together.

5. Given an n-variable map with a pair of adjacent 1-cells or 0-cells can result n-1 variable. Where as if a group of four adjacent sub cubes are formed than it can result n-2 variables. Finally if we have eight adjacent cells are grouped may result n-3 variable product or sum term.

Typical pair of sub cubes

w x z

1			
	1	1	
1		1	1
1			

ites4free.in

Typical group of four adjacent subcubes

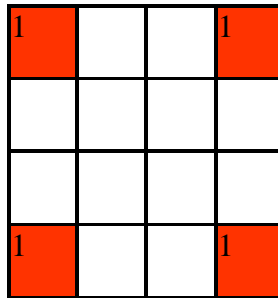
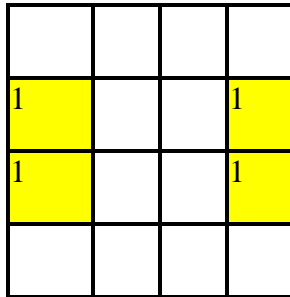
1	1		
1	1		

		1	
		1	
		1	
		1	

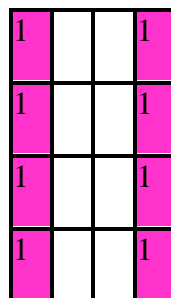
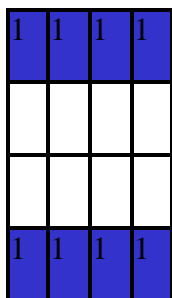
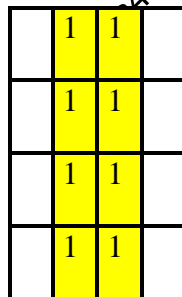
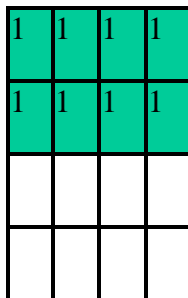
1	1	1	1



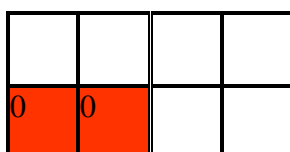
Typical group of four adjacent sub cubes.



Typical group of eight adjacent sub cubes.



Typical map sub cubes describing sum terms






0	0		
0	0		

		0	0
		0	0
		0	0
		0	0

**USING K-MAP TO OBTAIN MINIMAL EXPRESSION FOR COMPLETE BOOLEAN FUNCTIONS :**

How to obtain a minimal expression of SOP or POS of given function is discussed.

**PRIME IMPLICANTS and K-MAPS :**

CONCEPT OF ESSENTIAL PRIME IMPLICANT

00    01    11    10

0	0	0	1
0	0	1	1

$f(x,y,z) = xy + yz$

**ALGORITHM TO FIND ALL PRIME IMPLICANTS**

A General procedure is listed below

1. For an n-variable map make 2n entries of 1's. or 0's.

2. Assign  $I = n$ , so that find out biggest rectangular group with dimension  $2^a \times 2^b = 2^{n-1}$ .
3. If bigger rectangular group is not possible  $I = I-1$  form the subcubes which consist of all the previously obtained subcube repeat the step till all 1-cells or 0's are covered.

### Remaining is essential prime implicants

1. Essential prime implicants
2. Minimal sums
3. Minimal products

### MINIMAL EXPRESSIONS OF INCOMPLETE BOOLEAN FUNCTIONS

1. Minimal sums
2. Minimal products.

### EXAMPLE TO ILLUSTRATE HOW TO OBTAIN ESSENTIAL PRIMES

$$1. f(x,y,z) = \sum m(0,1,5,7)$$

$$\text{Ans } f(x,y,z) = xz + x y$$

$$2. f(w,x,y,z) = \sum m(1,2,3,5,6,7,8,13)$$

$$\text{Ans. } f(w,x,y,z) = w z + w y + xyz + w x y z$$

### MINIMAL SUMS

$$f(w,x,y,z) = \sum m(0,1,2,3,5,7,11,15)$$

### MINIMAL PRODUCTS

$$F(w,x,y,z) = \sum m(1,3,4,5,6,7,11,14,15)$$

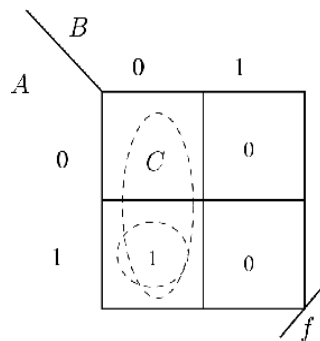
### MINIMAL EXPRESSIONS OF INCOMPLETE BOOLEAN FUNCTIONS

$$f(W,X,Y,Z) = \sum m(0,1,3,7,8,12) + dc(5,10,13,14)$$

### Entered-Variable K-Maps

A generalization of the k-map method is to introduce variables into the k-map squares. These are called *entered variable k-maps*. This is useful for functions of large numbers of variables, and can generally provide a clear way of representing Boolean functions.

An entered variable k-map is shown in Figure.

**Figure :** An entered variable k-map.

Note the variable  $C$  in the top left square. It corresponds to

$$\bar{A}.\bar{B}.C.$$

It can be looped out with the 1, since  $1=1+C$ , and we can loop out the two terms

$$\bar{A}.\bar{B}.C \text{ and } A.\bar{B}.C$$

to get

$$\bar{B}.C.$$

The remaining term

$$A.\bar{B}.\bar{C}$$

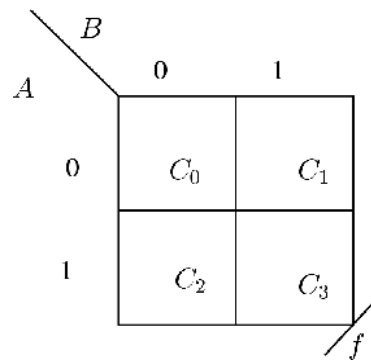
needs to be added to the cover, or more simply, just loop out the 1. The outcome is

$$f = A.\bar{B} + \bar{B}.C.$$

Figure shows another EV k-map, with four entered variables  $C_0, C_1, C_2, C_3$ . Each of these terms are different and must be looped out individually to get

$$f = \bar{A}.\bar{B}.C_0 + \bar{A}.B.C_1 + A.\bar{B}.C_2 + A.B.C_3.$$

**Figure:** Another entered variable k-map.



### Recommended question and answer - Unit-1

#### Jan-2009

1 a) Convert the given boolean function  $f(x, y, z) = [x + x Z (y + z)]$  into maxterm canonical formula and hence highlight the importance of canonical formul.1.

(5)

$$\begin{aligned}
 f(x,y,z) &= x (y + y) (z + z) \text{ ; - } x y z + x Z (y + y) \\
 &= x y z + x Y z + x y z + x Y z + x Y z + x Y z + x Y z + x Y z \\
 f(x, y, z) &= x y z + x Y z + x Y z + x Y z + x Y z + x Y z + x Y z + x Y z
 \end{aligned}$$

1 b) Distinguish the prime implicants and essential prime implicants. Determine the same of the function

$f(w, x, y, z) = I m(O, 1, 4, 5, -9, 11, 13, 15)$  using K-map and hence the minimal sum expression.

(5)

**Ans. :** After grouping the cells, sum terms which appear in the k-map are called prime implicants groups. It is observed than some cells may appear in only one prime implicant group, while other cells may appear in more than one prime implicants group. The cells which appear in only one prime implicant group are called essential cells and corresponding prime implicants are called essential prime implicants.

Fig. 1

$$f(w, x, y, z) = \bar{w} \bar{y} + yz + w z$$

**Jan-2008**

Q.1 a) Two motors M2 and M1; are controlled by three sensors S3, S2 and S1. One motor M2 is to run any time all three sensors are on. The other motor is to run whenever sensors S2 or S1 but not both are on and S3 is off. For all sensor combinations where M1 is on, M2 is to be off except when all the three sensors are off and then both motors must remain off. Construct the truth table and write the Boolean output equation.

(6)

Ans. :

Inputs			Output	
S <sub>3</sub>	S <sub>2</sub>	S <sub>1</sub>	M <sub>2</sub>	M <sub>1</sub>
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	0	0
1	0	0	0	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	0

Table 1 Truth table

Boolean output equation

$$M_2 = S_3 \Sigma m (S_2, S_1)$$

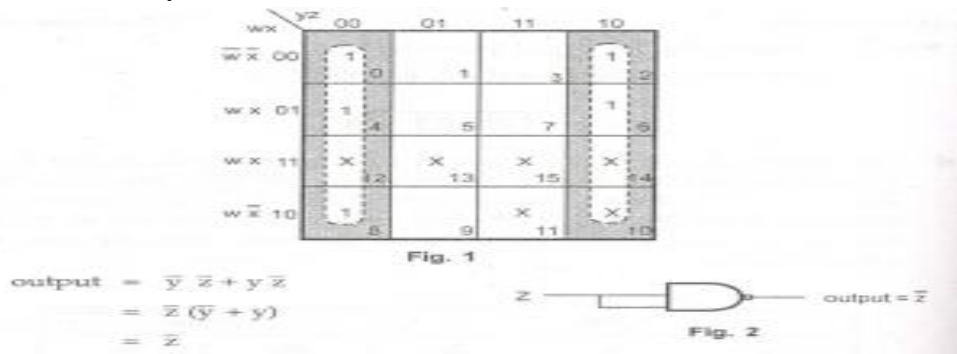
$$= S_3 \cdot S_2 \cdot S_1$$

$$M_1 = S_3 \cdot \bar{S}_2 \cdot S_1 + S_3 \cdot S_2 \cdot \bar{S}_1$$

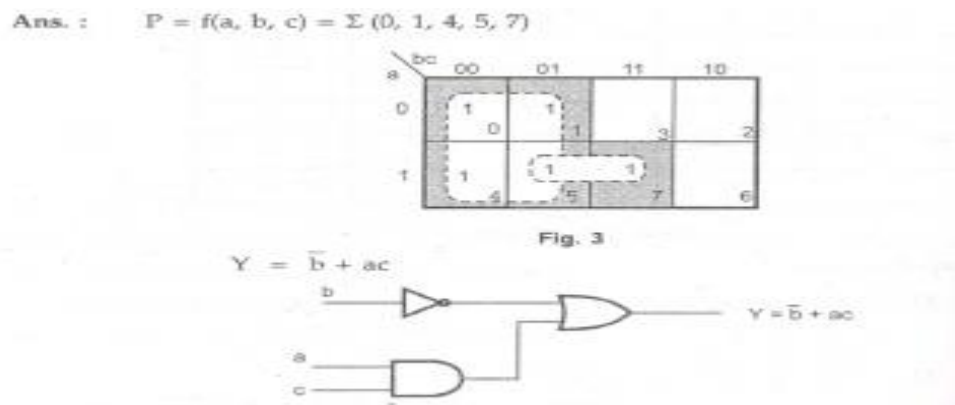
b) Simplify using Karnaugh map. Write the Boolean equation and realize using NAND gates

$$D = f(w, x, y, z) = L(0, 2, 4, 6, 8) + L d(10, 11, 12, 13, 14, 15). (6)$$

Ans. :  $D = f(w, x, y, z) = L(0, 2, 4, 6, 8) + Ld(10, 11, 12, 13, 14, 15)$ .

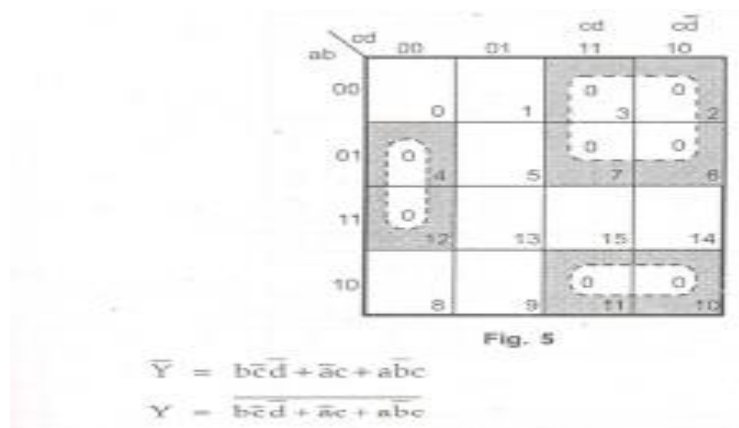


c. Simplify  $P = f(a, b, c) = L(0, 1, 4, 5, 7)$  using two variable Karnaugh map. Write the Boolean equation and realize using logic gates (8)



Q.2 a) Simplify using Karnaugh map  $L = f(a, b, c, d) = 1t(2, 3, 4, 6, 7, 10, 11, 12)$ . (6)

Ans. :  $L = f(a, b, c, d) = 1t(2, 3, 4, 6, 7, 10, 11, 12)$ .



**Aug 2009**

Q.1 a) Express the P.O.S. equations in a Maxterms list (decimal notations) form.

i)  $T = f(A, B, C) = (A + B + C)(A + B + \bar{C})(A + \bar{B} + C)$

ii)  $J = f(A, B, C, D) = (A + B + C + D) (A + B + C + D) (:4 + B + C + D)$   
 $(A + B + C + D) (A + B + C + D) (A + B + C + D) (4)$

i)  $T = f(A, B, C) = (A + B + C) (A + B + C) (A + B + C)$

...  $f(A, B, C) = M2 + M3 + M6 = 1t M(2, 3, 6)$

ii)  $J = f(A, B, C, D) = (A + B + C + D) (A + B + C + D) (A + B + C + D)$   
 $(A + B + C + D) (A + B + C + D) (A + B + C + D)$

$= M4 + Ms + Ms + M10 + M12 + M14$

$= 1t M(4, 5, 8, 10, 12, 14)$

b) Reduce the following function using K-map technique and implement using gates.

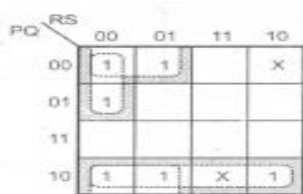
i)  $f(P, Q, R, S) = 1, m(0, 1, 4, 8, 9, 10) + d(2, 11)$

ii)  $f(A, B, C, D) = 1t m(0, 2, 4, 10, 11, 14, 15) (10)$

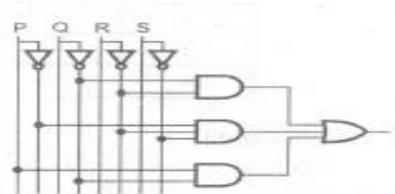
Ans.: i)  $(P, Q, R, S) 1, f(0, 1, 4, 8, 9, 10) + d(2, 11)$

$f(P, Q, R, S) QR + PRS + PQ$

ii)  $f(A, B, C, D) = \pi M(0, 2, 4, 10, 11, 14, 15)$

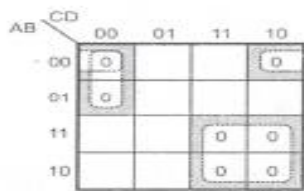


$f = \bar{Q}R + PRS + PQ$   
 (a) Simplification

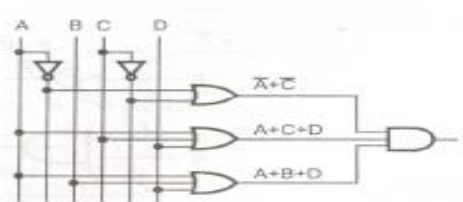


(b) Implementation

Fig. 1



$f = (\bar{A} + \bar{C})(A + C + D)(A + B + D)$   
 (a) Simplification



(b) Implementation

Fig. 2

c) Design a logic circuit with inputs P, Q, R so that output S is high whenever P is zero or whenever Q = R = 1. (6)

Ans.:

P	Q	R	S
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1

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

K-map simplification :

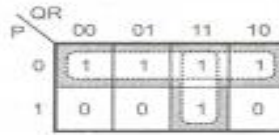


Fig. 3

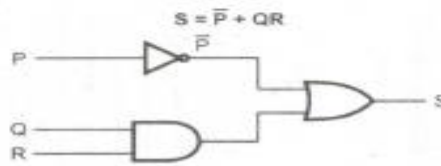


Fig. 4

notes4free.in

**Aug 2008**

Q.1 a) Simplify the following expression using Karnaugh map. Implement the simplified circuit using the gates as indicated.

i)  $f(ABCD) = \sum m(2, 3, 4, 5, 13, 15) + \sum l(8, 9, 10, 11)$  use only NAND gates

ii)  $f(ABCD) = \sum m(2, 3, 4, 6, 7, 10, 11, 12)$  use only NOR gates to implement these circuits.

i)  $f(ABCD) = \sum m(2, 3, 4, 5, 13, 15) + \sum l(8, 9, 10, 11)$

SOP = (Sum of product)

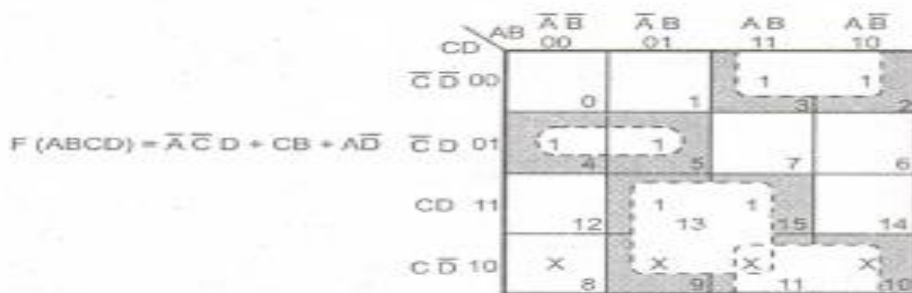


Fig. 1 (a)



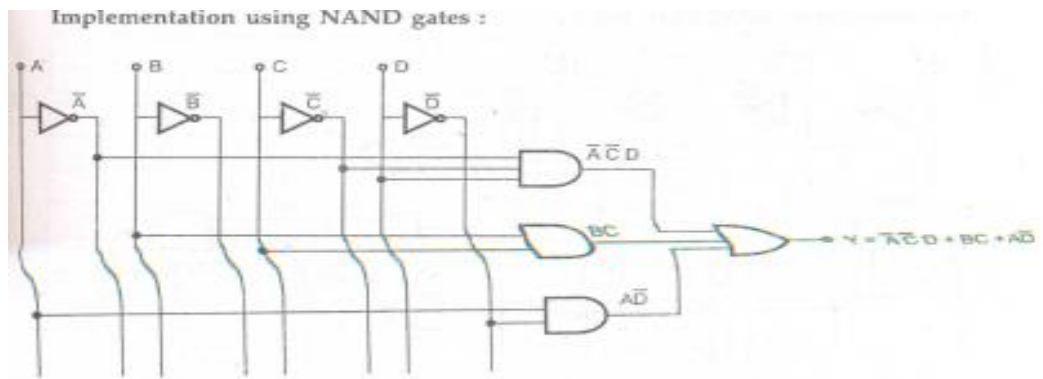


Fig. 1 (b)

ii)  $f(ABCD) = \Sigma \pi (2, 3, 4, 6, 7, 10, 11, 12)$

POS = (Product of sum)

K-map simplification :



Fig. 1 (c)

$$f(ABCD) = (C + \bar{A})(\bar{A} + D)(A + B + \bar{D})$$

$$= (\bar{A} + C)(\bar{A} + D)(A + B + \bar{D})$$

Implementation using NOR gates :

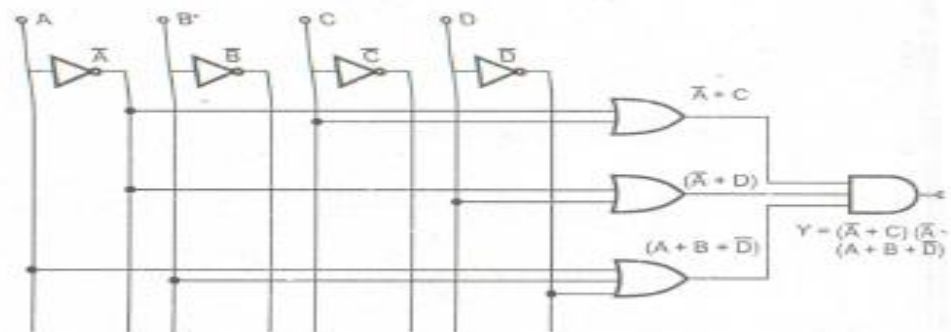


Fig. 1 (d)

Aug-2007

Q.1 a) Using the theorems of Boolean algebra, simplify the following.

i)  $y_1 = D(\overline{A+B}) + \overline{B}(C+AD)$

ii)  $y_2 = \overline{A}\overline{B} + ABC + A(B+\overline{A}\overline{B})$

iii)  $y_3 = \overline{A}\overline{B}\overline{C} + \overline{A}B\overline{C} + A\overline{B}\overline{C} + ABC$  [9]

Sol. : This topic is not included in the new syllabus.

b) Express the following expressions in canonical form :

i)  $y_1 = (A+B)(A+C)(B+\overline{C})$

ii)  $y_2 = AC + AB + BC$  [6]

Sol. : i) 
$$y_1 = (A+B)(A+C)(B+\overline{C})$$

$$= ((A+B) + (C \cdot \overline{C})) ((A+C) + (B \cdot \overline{B})) ((B+\overline{C}) + (A \cdot \overline{A}))$$

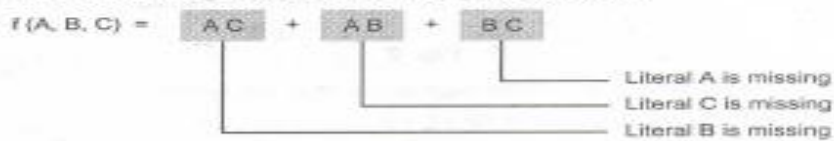
$$= (A+B+C)(A+B+\overline{C})(A+C+B)(A+C+\overline{B})$$

$$(B+\overline{C}+A)(B+\overline{C}+\overline{A})$$

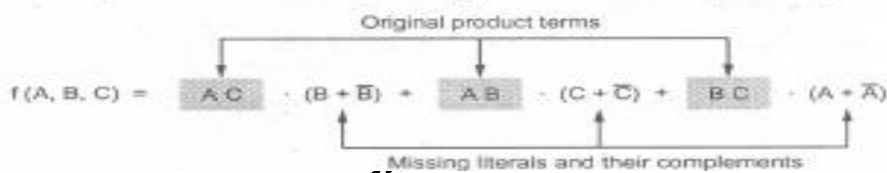
$$= (A+B+C)(A+B+\overline{C})(A+\overline{B}+C)(\overline{A}+B+\overline{C})$$

ii)  $y_2 = AC + AB + BC$

Step 1 : Find the missing literal/s in each product term



Step 2 : AND product term with (missing literal + its complement)



Step 3 : Expand the terms and reorder literals.

Expand :  $f(A, B, C) = ACB + AC\overline{B} + ABC + AB\overline{C} + BCA + BC\overline{A}$

Recorder :  $f(A, B, C) = ABC + A\overline{B}C + A\overline{B}\overline{C} + AB\overline{C} + ABC + \overline{A}BC$

Step 4 : Omit repeated product terms

$f(A, B, C) = ABC + A\overline{B}C + \overline{A}BC + AB\overline{C} + A\overline{B}\overline{C} + \overline{A}\overline{B}C$

$\therefore f(A, B, C) = ABC + A\overline{B}C + \overline{A}BC + AB\overline{C} + A\overline{B}\overline{C} + \overline{A}\overline{B}C$

Sol. : The Boolean expression for EX-OR gate is :  $Y = A\overline{B} + \overline{A}B$

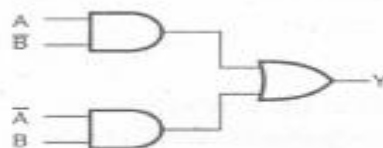


Fig. 1

We can implement AND-OR logic by using NAND-NAND logic as shown in Fig. 2.

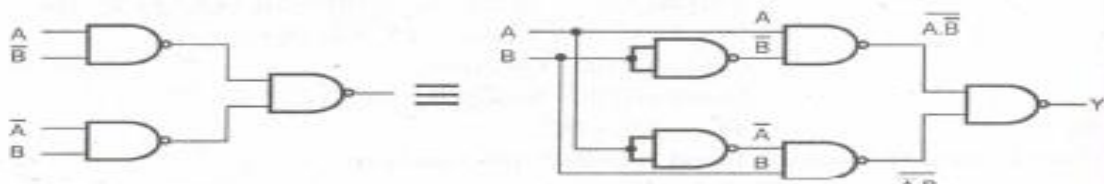


Fig. 2

Q.2 a) i) Express the following min term expression in POS form :

$$y(A, B, C, D) = \sum m(1, 3, 5, 6, 7, 9, 10, 12, 15)$$

ii) Express the following max term expression in SOP form :

$$y(A, B, C) = \pi M(0, 3, 5, 6)$$

Sol. :  $y(A, B, C, D) = \sum m(1, 3, 5, 6, 7, 9, 10, 12, 15)$

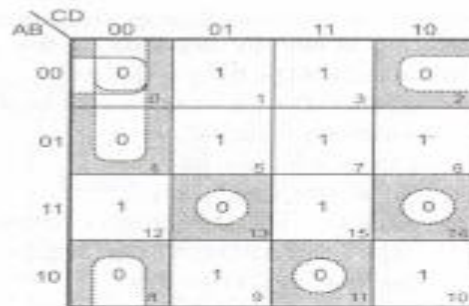


Fig. 3

$$\begin{aligned} \bar{Y} &= \bar{A} \bar{B} \bar{C} + \bar{A} \bar{C} \bar{D} + \bar{B} \bar{C} \bar{D} + A \bar{B} \bar{C} D + A \bar{B} C \bar{D} + A \bar{B} C D \\ &= (A + B + D) (A + C + D) (B + C + D) (\bar{A} + \bar{B} + C + \bar{D}) \\ &\quad (\bar{A} + \bar{B} + \bar{C} + D) (\bar{A} + B + \bar{C} + \bar{D}) \end{aligned}$$

ii)  $Y(A, B, C) = \pi M(0, 3, 5, 6)$

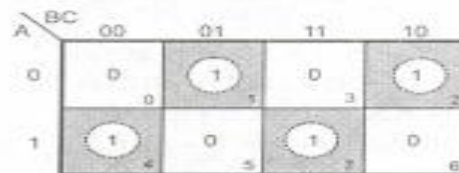


Fig. 4

$$Y = \bar{A} \bar{B} \bar{C} + \bar{A} \bar{B} C + A \bar{B} \bar{C} + A \bar{B} C$$

b) i) What are the advantage, disadvantages of K map?

ii) Simplify the following function in SOP form using K Map:

$$f(A, B, C, D) = \bar{A} \bar{B} \bar{C} + A \bar{D} + B \bar{D} + C \bar{D} + A \bar{C}$$

Sol. : i) **Advantages of K-map method:**

1. It provides a systematic approach for simplifying a Boolean expression.
2. It is very convenient method for simplifying a Boolean expression upto six variables.

**Disadvantages of K-map method:**

1. As the number of variables increases it is difficult to make judgements about which combinations form the **minimum** expression. In case of complex problem with 7, 8, or even 10 variables it is almost an impossible task to simplify expression by the mapping method.

2. Another important point is that the K-map simplification is manual technique and simplification process is heavily depends on the human abilities.

$$\begin{aligned}
 \text{ii) } f(A, B, C, D) &= \overline{A} \overline{B} C + A D + B \overline{D} + C \overline{D} + A \overline{C} \\
 &= (\overline{A} \overline{B} C) (D + \overline{D}) + (A D) + (B + \overline{B}) (C + \overline{C}) \\
 &\quad + B \overline{D} (A + \overline{A}) (C + \overline{C}) + \\
 &\quad C \overline{D} (A + \overline{A}) (B + \overline{B}) \\
 &\quad + A \overline{C} (B + \overline{B}) (D + \overline{D}) \\
 &= \overline{A} \overline{B} C D + \overline{A} \overline{B} C \overline{D} + (A B D + A \overline{B} D) \\
 &\quad (C + \overline{C}) + (A B \overline{D} + \overline{A} B \overline{D}) (C + \overline{C}) + \\
 &\quad (A C \overline{D} + \overline{A} C \overline{D}) (B + \overline{B}) + \\
 &\quad (A B \overline{C} + \overline{A} B \overline{C}) (D + \overline{D}) \\
 &= \overline{A} \overline{B} C D + \overline{A} \overline{B} C \overline{D} + A B C D + \\
 &\quad \overline{A} B C D + \overline{A} B \overline{C} D + \overline{A} B C \overline{D} + \\
 &\quad \overline{A} B C D + \overline{A} B C \overline{D} + A B C \overline{D} + A B C D \\
 &\quad + A B C \overline{D} + \overline{A} B C D + A C B D + \\
 &\quad \overline{A} B C D + \overline{A} B C \overline{D} + \overline{A} B \overline{C} D + \\
 &\quad \overline{A} B C \overline{D} + \overline{A} B \overline{C} \overline{D} \\
 &= A B C D + \overline{A} \overline{B} C D + \overline{A} B \overline{C} D + \\
 &\quad \overline{A} B C D + A B \overline{C} D + A B C \overline{D} + \\
 &\quad A B C D + A B C \overline{D} + A B C D + \\
 &\quad \overline{A} B C \overline{D} + \overline{A} B C D + \overline{A} B \overline{C} D
 \end{aligned}$$

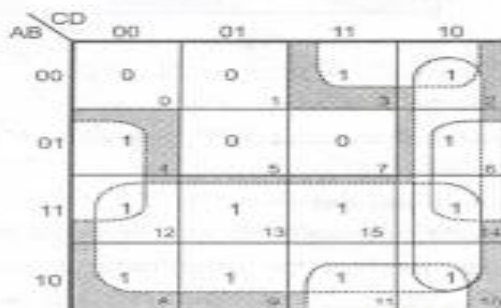


Fig. 5

$$f(A, B, C, D) = A + C\overline{D} + \overline{B}C + B\overline{D}$$

c) Simplify the following function in POS form using K map:  
 $f(A, B, C, D) = \pi M(0, 1, 2, 5, 8, 9, 10)$ .

$$f(A, B, C, D) = \pi M(0, 1, 2, 5, 8, 9, 10)$$



Fig. 6

$$\begin{aligned}
 \overline{f} &= \overline{B} \overline{C} + B \overline{D} + A C D \\
 f &= \overline{(\overline{B} \overline{C} + B \overline{D} + A C D)} \\
 &= (\overline{\overline{B} \overline{C}}) (\overline{B \overline{D}}) (\overline{A C D}) \\
 &= (\overline{B + \overline{C}}) (\overline{B + \overline{D}}) (\overline{A \overline{C} + D}) \\
 &= (B + C) (B + D) ((\overline{A + \overline{C}}) + \overline{D}) \\
 &= (B + C) (B + D) (A + C + \overline{D})
 \end{aligned}$$

d) Simplify the following function in SOP form using K map :  
 $y(w, x, y, z) = \sum m(1, 2, 3, 5, 9, 12, 14, 15) + \sum d(4, 8, 11)$

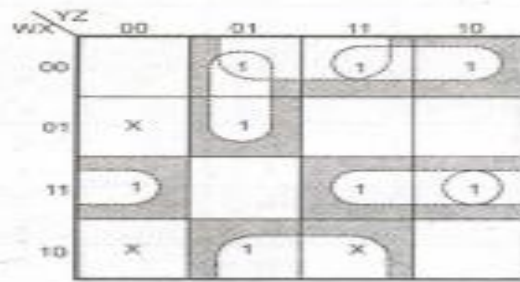


Fig. 7

$$Y = \bar{w} \bar{x} y + \bar{x} z + \bar{w} \bar{y} z + wx\bar{z} + wxy$$

notes4free.in

**Principles of combinational Logic-2**

Quine-McCluskey minimization technique- Quine-McCluskey using don't care terms, reduced Prime Implicant Tables, Map entered variables

**Recommended readings:**

1. John M Yarbrough, "Digital Logic Applications and Design", Thomson Learning, 2001.

Unit- 3.5, 3.6

notes4free.in

**QUINE – McCLUSKEY METHOD**

Using K-maps for simplification of Boolean expressions with more than six variables becomes a tedious and difficult task. Therefore a tabular method illustrate below can be used for the purpose.

**ALGORITHM FOR GENERATING PRIME IMPLICANTS**

The algorithm procedure is listed below

1. Express each minterm of the function in its binary representation.
2. List the minterms by increasing index.
3. Separate the sets of minterms of equal index with lines.
4. Let  $i = 0$ .
5. Compare each term of index  $I$  with each term of index  $I+1$ . For each pair of terms that can combine which has only one bit position difference.
6. Increase  $I$  by 1 and repeat step 5. The increase of  $I$  continued until all terms are compared. The new list containing all implicants of the function that have one less variable than those implicants in the generating list.
7. Each section of the new list formed has terms of equal index. Steps 4,5, and 6 are repeated on this list to form another list. Recall that two terms combine only if they have their dashes in the same relative positions and if they differ in exactly one bit position.
8. The process terminates when no new list is formed.
9. All terms without check marks are prime implicants.

Example: Find all the prime implicants of the function

$$f(w,x,y,z) = \sum m(0,2,3,4,8,10,12,13,14)$$

Step 1: Represent each minterm in its 1-0 notation

no.	minterm	1-0 notation	index
0	w x y z	0 0 0 0	0
2	w x y z	0 0 1 0	1
3	w x y z	0 0 1 1	2
4	w x y z	0 1 0 0	1
8	w x y z	1 0 0 0	1
10	w x y z	1 0 1 0	2
12	w x y z	1 1 0 0	2
13	w x y z	1 1 0 1	3
14	w x y z	1 1 1 0	3

Step 2: List the minterm in increasing order of their index.

No.	w x y z	index
0	0 0 0 0	Index 0
2	0 0 1 0	
4	0 1 0 0	Index 1
8	1 0 0 0	
3	0 0 1 1	
10	1 0 1 0	Index 2
12	1 1 0 0	
13	1 1 0 1	
14	1 1 1 0	Index 3



	W x y z	index
0,2	0 0 - 0	
0,4	0 - 0 0	
0,8	- 0 0 0	
2,3	0 0 1 -	
2,10	- 0 1 0	
	- 1 0 0	
4,12	1 0 - 0	
8,10	1 - 0 0	
8,12	1 - 1 0	
10,14	1 1 0 -	
12,13	1 1 - 0	
12,14		

notes4free.in

	w x y z
(0, 2, 8, 10)	__ 0 __ 0
(0, 4, 8,12 )	__ __ 0 0(index 0)
(8,10,12,14)	1__ __ 0 (index 1)

$$F(w,x,y,z)=x z + y z +w z+w x y +w x z$$

**PETRICK'S METHOD OF DETERMINING IRREDUNDANT EXPRESSIONS****FIND THE PRIME IMPLICANTS AND IRREDUNDANT EXPRESSION**

$$F(W,X,Y,Z) = \sum M(0,1,2,5,7,8,9,10,13,15)$$

$$A = X Y, B = X Z, C = Y Z, D = X Z$$

$$P = (A+B)(A+C)(B)(C+D)(D)(A+B)(A+C)(B)(C+D)(D)$$

$$P = (A+C)(BD) = ABD + BCD$$

$$F1(W,X,Y,Z) = ABD = X Y + X Z + X Z$$

$$F2(W,X,Y,Z) = BCD = X Z + Y Z + X Z$$

notes4free.in

**DECIMAL METHOD FOR OBTAINING PRIME IMPLICANTS**

The prime implicants can be obtained for decimal number represented minterms. In this procedure binary number are not used to find out prime implicants

$$f(w, x, y, z) = \sum m(0,5,6,7,9,10,13,14,15)$$

$$f_{sop} = xy + xz + xyz + wyz + w x y z$$

**MAP ENTERED VARIABLE (MEV)**

It is graphical approach using k-map to have a variable of order n. Where in we are using a K-map of n-1 variable while map is entered with output function and variable.

$$f(w,x,y,z) = \sum m(2,3,4,5,13,15) + dc(8,9,10,11)$$

Ans. fsop =  $wz + xy + wx'y$

- ❖ karnaugh mapping is the best manual technique for boolean equation simplification, yet when the map sizes exceed five or six variable unwidely.
- ❖ the technique called “map entered variables “ ( mevs ) increases the effective size of a karnaugh maps, allowing a smaller map to handle a greater no. of variables
- ❖ the map dimension and the no. of problem variables are related by  $2^n = m$ , where  $n$  = no.of problem variable,  $m$  = no. of squares in k-maps. mev k-maps permit a cell to contain a single (x) or a complete switching expression, in addition to the 1s, 0s and don't care terms.

**STEP:1**

✓ IF THE OUTPUT VARIABLE IS A “0” FOR BOTH STANDARD MINTERM COVERED BY MEV MAP SQUARE, THEN A “0” IS WRITTEN IN THAT MEV MAP SQUARE.

**STEP:2**

✓ IF THE OUTPUT VARIABLE IS “1” FOR BOTH STANDARD MINTERM COVERED BY MEV MAP SQUARE, THEN A “1” IS WRITTEN IN THAT MEV MAP SQUARE.

**STEP:3**

✓ IF FOR THE MINTERMS COVERED BY THE MEV MAP SQUARE , THE OUTPUT VARIABLE HAS THE SAME VALUE AS THE MEV.

**STEP:4**

✓ IF FOR THE STANDARD MINTERM COVERED BY A MEV MAP SQUARE, THE OUTPUT & MEV VARIABLES ARE COMPLIMENTS WRITE THE MEV COMPLEMENT INTO MEV MAP SQUARE.

**STEP:5**

✓ IF FOR STANDARD MINTERMS COVERED BY A MEV MAP SQUARE, THE OUTPUT VARIABLE IS A DON'T CARE TERM, WRITE A DON'T CARE SYMBOL "x" INTO THE MEV MAP SQUARE.

**STEP:6**

✓ IF FOR STANDARD MINTERMS COVERED BY A MEV MAP SQUARE, THE OUTPUT VARIABLE IS A DON'T CARE TERM IN 1 CASE & "0" IN THE OTHER, WRITE "0"

✓ IF THE OUTPUT VARIABLE IS A DON'T CARE TERM IN 1 CASE & "1" IN THE OTHER, WRITE "1"

TO READ THE SIMPLIFIED FUNCTION FROM A MEV K-MAP, FOLLOW THESE STEPS:

**STEP:1**

✓ DETERMINE THE EPIs CONSISTING OF ONLY 1s ALONG WITH ANY DON'T CARE TERMS THAT MAY EXIST. { i.e. COVER THE 1s IN K-MAP. }

**STEP:2**

✓ CONSIDER THE 1s AS DON'T CARE TERMS ONCE STEP 1 IS COMPLETED, BECAUSE ALL OF THE 1s HAVE BEEN PREVIOUSLY COVERED.

**STEP:3**

✓ GROUP ALL THE IDENTICAL MEV TERMS WITH 1s OR DON'T CARE TERMS TO MAXIMIZE THE MEV EPI SIZE.

✓ ANY MINTERM THAT ARE NOT CONTAINED IN THE MEV EPI ARE CONSIDERED TO BE 0s. { i.e. COVER ALL THE MEVs IN THE K-MAP }

**STEP:4**

✓ DETERMINE THE MEV EPIs BY READING THE K-MAP IN THE NORMAL FASHION.

✓ THEN " AND " THE MEV VARIABLE OR EXPRESSION WITH THE REMAINING MAP VARIABLES.

**Recommended question and answer –unit-2**

**Jan-2009**

Q.2 a) Using Quine-Mcluskey method and prime implicant reduction table, obtain the minimal sum expression for the Boolean function

$$F(w, x, y, z) = \sum m(0, 4, 6, 7, 8, 9, 10, 11, 15) \dots \tag{12}$$

$$f(w, X, y, z) = \sum m(0, 4, 6, 7, 8, 9, 10, 11, 15)$$

Minterms	Binary representation	Minterms	Binary representation
m <sub>1</sub>	0 0 0 1	m <sub>1</sub>	0 0 0 1 ✓
m <sub>4</sub>	0 1 0 0	m <sub>4</sub>	0 1 0 0 ✓
m <sub>6</sub>	0 1 1 0	m <sub>6</sub>	1 0 0 0 ✓
m <sub>7</sub>	0 1 1 1	m <sub>6</sub>	0 1 1 0 ✓
m <sub>8</sub>	1 0 0 0	m <sub>9</sub>	1 0 0 1 ✓
m <sub>9</sub>	1 0 0 1	m <sub>10</sub>	1 0 1 0 ✓
m <sub>10</sub>	1 0 1 0	m <sub>7</sub>	0 1 1 1 ✓
m <sub>11</sub>	1 0 1 1	m <sub>11</sub>	1 0 1 1 ✓
m <sub>15</sub>	1 1 1 1	m <sub>15</sub>	1 1 1 1

Minterms	Binary representation	Minterms	Binary representation
1, 9	_ 0 0 1	1, 9	_ 0 0 1
4, 6	0 1 _ 0	4, 6	0 1 _ 0
8, 9	1 0 0 _ ✓	8, 9, 10, 11	1 0 _ _
8, 10	1 0 _ 0 ✓	6, 7	0 1 1 _
6, 7	0 1 1 _	7, 15	_ 1 1 1
9, 11	1 0 _ 1 ✓	11, 15	1 _ 1 1
10, 11	1 0 1 _ ✓		
7, 15	_ 1 1 1		
11, 15	1 _ 1 1		

$$\therefore f(w, x, y, z) = x y z + W x Z + W x + W x Y + x Y z + W Y z$$

**b) Obtain the minimal product of the following Boolean functions using VEM technique:**

$$f(w, x, y, z) = \sum m(1, 5, 7, 10, 11) + dc(2, 3, 6, 13) \dots \tag{8}$$

**Ans. :**  $f(w, x, y, z) = \sum m(1, 5, 7, 10, 11) + dc(2, 3, 6, 13)$

Writing these minterms in SOP form we get,

$$f(w, x, y, z) = \bar{w} \bar{x} \bar{y} z + \bar{w} \bar{x} y \bar{z} + \bar{w} \bar{x} y z + \bar{w} x \bar{y} z + \bar{w} x y \bar{z} + \bar{w} x y z + w \bar{x} y \bar{z} + w \bar{x} y z + w x \bar{y} z$$

Now converting 4-variable truth table into 3-variable truth table we get,

$$f(w, x, y, z) = \text{ill}a z + \text{ill}1 Z + \text{ill}l Z + \text{ill}2 Z + \text{ill}3 Z + \text{ill}3 Z + \text{ill}S Z + \text{ill}S Z + \text{ill}6 Z$$

$$= \text{ill}a Z + \text{ill}l (z + z) + \text{ill}2 Z + \text{ill}3 (z + z) + \text{ill}S (z + z) + \text{ill}6 Z$$

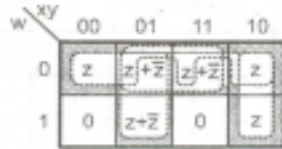


Fig. 2

Applying grouping technique we get,

$$f(w, x, y, z) = w Z + X Y + w Y + x y Z$$

**Jan-2008**

c. Simplify  $P = f(a, b, c) = L(0,1, 4, 5, 7)$  using two variable Karnaugh map. Write the Boolean equation and realize using logic gates (8)

Ans. :  $P = f(a, b, c) = \Sigma(0, 1, 4, 5, 7)$

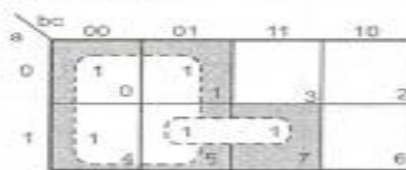
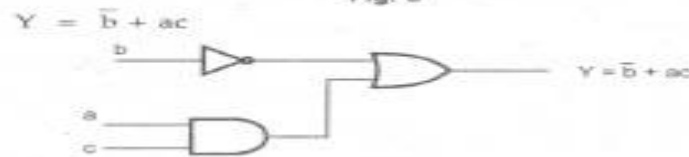


Fig. 3



b) Simplify using Quine Mc Cluskey tabulation algorithm -

$$v = lea, b, c, d) = L(2, 3, 4, 5, 13, 15) + L d(8, 9, 10, 11) \quad (14)$$

Ans:

Step-1 : List all minterms in binary form.

Minterms	Binary representation			
$m_2$	0	0	1	0
$m_3$	0	0	1	1
$m_4$	0	1	0	0
$m_5$	0	1	0	1
$m_{13}$	1	1	0	1
$m_{15}$	1	1	1	1
$dm_8$	1	0	0	0
$dm_9$	1	0	0	1
$dm_{10}$	1	0	1	0
$dm_{11}$	1	0	1	1

Step-2 : Arrange the minterms according to number of 1's.

Minterms	Binary representation
$m_2$ →	0 0 1 0 ✓
$m_4$ →	0 1 0 0 ✓
$m_8$ →	1 0 0 0 ✓
$m_3$	0 0 1 1 ✓
$m_5$	0 1 0 1 ✓
$m_9$	1 0 0 1 ✓
$m_{10}$	1 0 1 0 ✓
$m_{13}$	1 1 0 1
$m_{11}$	1 0 1 1
$m_{15}$	1 1 1 1

Step-3 :

Minterm	Binary representation
2, 3	0 0 1 - ✓
2, 10	- 0 1 0 ✓
4, 5	0 1 0 -
8, 10	1 0 - 0 ✓
8, 9	1 0 0 - ✓
3, 11	- 0 1 1 ✓
5, 13	- 1 0 1
9, 13	1 - 0 1 ✓
9, 11	1 0 - 1 ✓
10, 11	1 0 1 - ✓
13, 15	1 1 - 1 ✓
11, 15	1 - - 1 ✓

Step-4 :

Minterms				Binary representation			
2,	3,	10,	11	-	0	1	-
2,	10,	3	11	-	0	1	-
8,	10,	9	11	1	0	-	-
8,	9,	10,	11	1	0	-	-
9,	13,	11,	15	1	-	-	1
9,	11,	13,	15	1	-	-	1

Step-5 :

Prime implicants			Binary representation					
$\bar{A}$	$\bar{B}$	$\bar{C}$	4, 5	→	0	1	0	-
$\bar{B}$	$\bar{C}$	$\bar{D}$	5, 13	→	-	1	0	1
$\bar{B}$	$\bar{C}$		2, 3, 10, 13	→	-	0	1	-
$\bar{B}$	$\bar{C}$		2, 10, 3, 13	→	-	0	1	-
$A$	$\bar{B}$		8, 10, 9, 11	→	1	0	-	-
$A$	$\bar{B}$		8, 9, 10, 11	→	1	0	-	-
$A$	$\bar{D}$		9, 13, 11, 15	→	1	-	-	1
$A$	$\bar{D}$		9, 11, 13, 15	→	1	-	-	1

Step-6 :

Prime implicants	$m_2$	$m_3$	$m_4$	$m_5$	$m_{13}$	$m_{15}$	$m_8$	$m_9$	$m_{10}$	$m_{11}$
$\bar{A} \bar{B} \bar{C}$ 4, 5			⊙	⊙						
$\bar{B} \bar{C} \bar{D}$ 5, 13				⊙	⊙					
$\bar{B} \bar{C}$ 2, 3, 10, 13	⊙	⊙			⊙				⊙	
$\bar{B} \bar{C}$ 2, 10, 3, 13	⊙	⊙			⊙				⊙	
$A \bar{B}$ 8, 10, 9, 11							⊙	⊙	⊙	⊙
$A \bar{B}$ 8, 9, 10, 11							⊙	⊙	⊙	⊙
$A \bar{D}$ 9, 13, 11, 15					⊙	⊙		⊙		⊙
$A \bar{D}$ 9, 11, 13, 15					⊙	⊙		⊙		⊙

Final expression

$$Y = \bar{A}\bar{B}\bar{C} + \bar{B}\bar{C}\bar{D} + \bar{B}\bar{C} + A\bar{B} + A\bar{D}$$

Aug-2009

Q.2 a) Using Quine McCluskey method simplify the following function.

$$f(a, b, c, d) = \sum m(0, 1, 2, 3, 8, 9)$$

Ans. :

Minterms	Binary Representation				Minterms	Binary Representation			
$m_0$	0	0	0	0	$m_0$	0	0	0	0
$m_1$	0	0	0	1	$m_1$	0	0	0	1
$m_2$	0	0	1	0	$m_2$	0	0	1	0
$m_3$	0	0	1	1	$m_8$	1	0	0	0
$m_8$	1	0	0	0	$m_3$	0	0	1	1
$m_9$	1	0	0	1	$m_9$	1	0	0	1



Minterms	Binary Representation	Minterms	Binary Representation
0,1 ✓	0 0 0 -	0, 1, 8, 9	- 0 0 -
0,2 ✓	0 0 - 0	0, 1, 2, 3	0 0 - -
0,8 ✓	- 0 0 0		
1,3 ✓	0 0 - 1		
1,9 ✓	- 0 0 1		
2,3 ✓	0 0 1 -		
8,9 ✓	1 0 0 -		

Prime Implicants	Binary Representation
0, 1, 8, 9	- 0 0 - ( $\bar{B} \bar{C}$ )
0, 1, 2, 3	0 0 - - ( $\bar{A} \bar{B}$ )

$\therefore \sum m(0, 1, 2, 3, 8, 9) = \bar{B} \bar{C} + \bar{A} \bar{B}$

b) Write the map entered variable K-map for the Boolean function.

$f(w \sim X, y, z) = Lm(2, 9, 10, 11, 13, 14, 15)$

$f(w, X, y, z) = Lm(2, 9, 10, 11, 13, 14, 15)$

notes4free.in

Minterms in Decimal	Minterms in Binary				f	Entry in MEV map
	w	x	y	z (MEV)		
0 { 0 1	0 0	0 0	0 0	0 1	0 0	{ 0 0
1 { 2 3	0 0	0 0	1 1	0 1	1 0	{ 1 z
2 { 4 5	0 0	1 1	0 0	0 1	0 0	{ 0 0
3 { 6 7	0 0	1 1	1 1	0 1	0 0	{ 0 0
4 { 8 9	1 1	0 0	0 0	0 1	0 1	{ 0 z

5	$\begin{cases} 10 \\ 11 \end{cases}$	$\begin{cases} 1 \\ 1 \end{cases}$	$\begin{cases} 0 \\ 0 \end{cases}$	$\begin{cases} 1 \\ 1 \end{cases}$	$\begin{cases} 0 \\ 1 \end{cases}$	$\begin{cases} 1 \\ 1 \end{cases}$	$\left. \vphantom{\begin{matrix} 1 \\ 1 \end{matrix}} \right\} z$
6	$\begin{cases} 12 \\ 13 \end{cases}$	$\begin{cases} 1 \\ 1 \end{cases}$	$\begin{cases} 1 \\ 1 \end{cases}$	$\begin{cases} 0 \\ 0 \end{cases}$	$\begin{cases} 0 \\ 1 \end{cases}$	$\begin{cases} 0 \\ 1 \end{cases}$	$\left. \vphantom{\begin{matrix} 0 \\ 1 \end{matrix}} \right\} z$
7	$\begin{cases} 14 \\ 15 \end{cases}$	$\begin{cases} 1 \\ 1 \end{cases}$	$\begin{cases} 1 \\ 1 \end{cases}$	$\begin{cases} 1 \\ 1 \end{cases}$	$\begin{cases} 0 \\ 1 \end{cases}$	$\begin{cases} 1 \\ 1 \end{cases}$	$\left. \vphantom{\begin{matrix} 1 \\ 1 \end{matrix}} \right\} z$

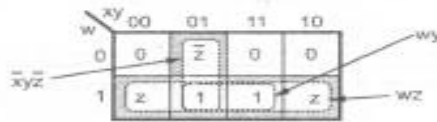


Fig. 5

$\therefore f(w, x, y, z) = \bar{x}y\bar{z} + wy + wz$

**Aug-2008**

**Q.2a)** Simplify the logic function given below, using Quine-McCluskey technique.  $Y(ABCD) = \sum m(0, 1, 3, 7, 8, 9, 11, 15)$ . Realize the expression using universal gates.

$Y(ABCD) = \sum m(0, 1, 3, 7, 8, 9, 11, 15)$

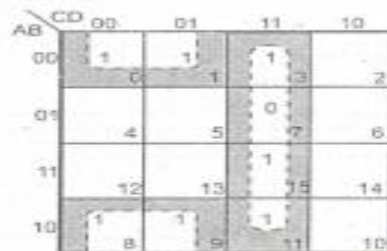


Fig. 3 (a)

$Y = CD + \bar{B}\bar{C}$

Simplification using Quine-McCluskey method :

Gr	Minterm	Representation in binary form			
		A	B	C	D
1	$m_0$	0	0	0	0
2	$m_1$	0	0	0	1
	$m_8$	1	0	0	0
3	$m_3$	0	0	1	1
	$m_9$	1	0	0	1
4	$m_7$	0	1	1	1
	$m_{11}$	1	0	1	1
5	$m_{15}$	1	1	1	1

Combination of minterms into general of two :

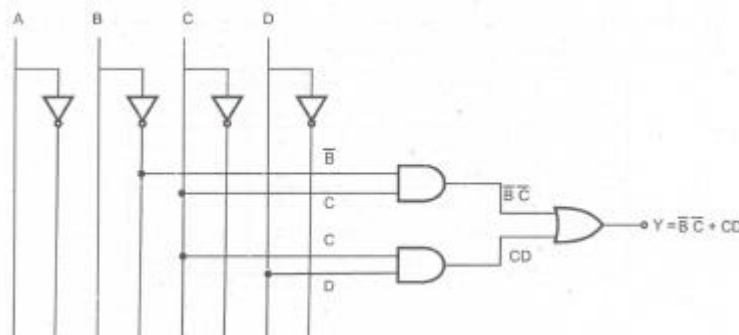
Gr	Minterms	Binary representation			
		A	B	C	D
0	$m_0 - m_1$	0	0	0	-
	$m_0 - m_8$	-	0	0	0
1	$m_1 - m_3$	0	0	-	1
	$m_1 - m_9$	-	0	0	1
	$m_8 - m_9$	1	0	0	-

2	$m_3 - m_7$	0	-	1	1
	$m_3 - m_{11}$	-	0	1	1
	$m_9 - m_{11}$	1	0	-	1
3	$m_7 - m_{15}$	-	1	1	1
	$m_{11} - m_{15}$	1	-	1	1

Gr.	Minterms	Binary representation			
		A	B	C	D
0	$m_0 - m_1 - m_8 - m_9$	-	0	0	-
	$m_0 - m_8 - m_1 - m_9$	-	0	0	-
1	$m_1 - m_3 - m_9 - m_{11}$	-	0	-	1
	$m_1 - m_9 - m_3 - m_{11}$	-	0	-	1
2	$m_3 - m_7 - m_{11} - m_{15}$	-	-	1	1
	$m_3 - m_{11} - m_7 - m_{15}$	-	-	1	1

Prime implicants	Decimal no.	Given minterms							
		0	1	3	7	8	9	11	15
		(X)	X						
$\overline{B} \overline{C}$	0, 1, 8, 9	(X)	X			(X)	X		
$\overline{B} D$	1, 3, 9, 11		X	X			X		
CD	3, 7, 11, 15			X	(X)			X	(X)

$Y(ABCD) = \overline{B} \overline{C} + CD$



b) Simplify the logic function given below using variable - entered mapping (VEM)

technique.  $Y (ABeD) = L m (1, 3, 4, 5, 8, 9, 10, 15) + L d (2, 7, 11, 12, 13).$  (8)

Ans. :

1. Use A, B, C as ordinary K-map variable
2. Make D the map-entered variable

A	B	C	D	f	Map Entry
0	0	0	0	0	} 0
0	0	0	1	1	
0	0	1	0	X	} 0 + $\bar{D}X$
0	0	1	1	1	
0	1	0	0	1	} 1
0	1	0	1	1	
0	1	1	0	0	} D
0	1	1	1	X	
1	0	0	0	1	} 1
1	0	0	1	1	
1	0	1	0	1	} $\bar{D} + DX$
1	0	1	1	X	
1	1	0	0	0	} 0
1	1	0	1	X	
1	1	1	0	0	} D
1	1	1	1	1	

3. Now using VEM, make the maps :

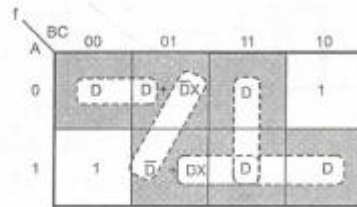


Fig. 4

$$\begin{aligned}
 f &= \bar{A} \bar{B} + BC + AB + AC + \bar{B}C \\
 &= \bar{A} \bar{B} + AB + BC + \bar{B}C + AC \\
 f &= \bar{A} \bar{B} + AB + C + (B + \bar{B}) + AC \\
 f &= \bar{A} \bar{B} + AB + C + AC \\
 &= \bar{A} \bar{B} + AB + C (1 + A) \\
 f &= \bar{A} \bar{B} + AB + C
 \end{aligned}$$

Module -2

Hr:10

**Analysis and design of combinational logic - I:**

General approach, Decoders-BCD decoders, Encoders

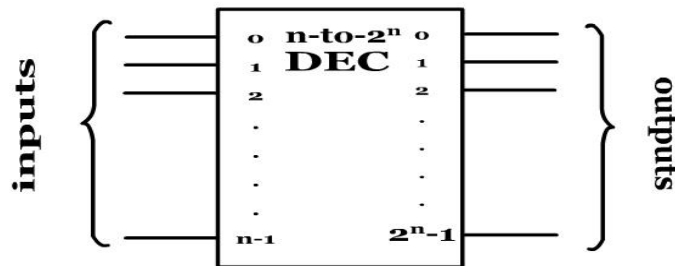
**Recommended readings:**

1. John M Yarbrough, “Digital Logic Applications and Design”, Thomson Learning, 2001.

**Unit- 4.1, 4.3, 4.4**

### Decoder

A Decoder is a multiple input ,multiple output logic circuit.The block diagram of a decoder is as shown below.

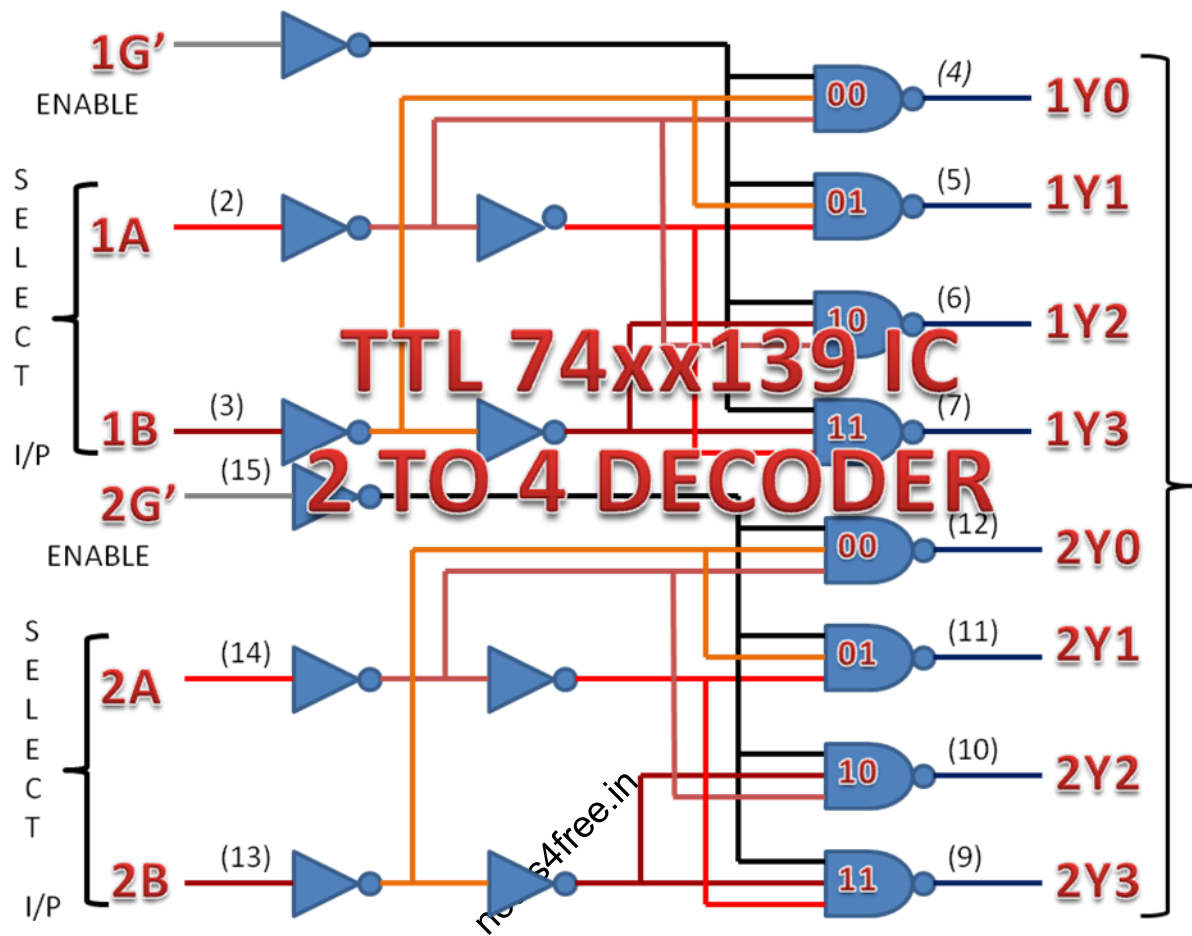


**An n-to-2<sup>n</sup>line decoder symbol.**

notes4free.in

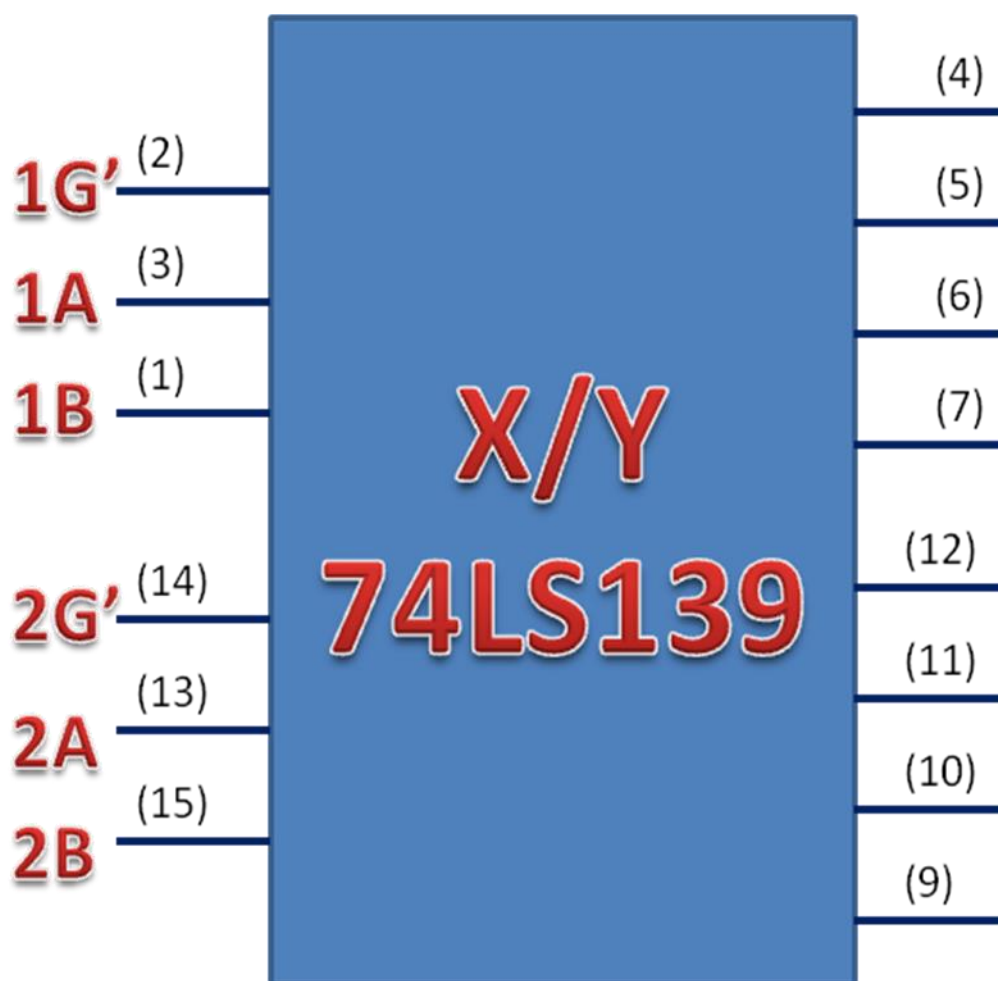
The most commonly used decoder is a n –to 2<sup>n</sup> decoder which ha n inputs and 2<sup>n</sup>

Output lines .



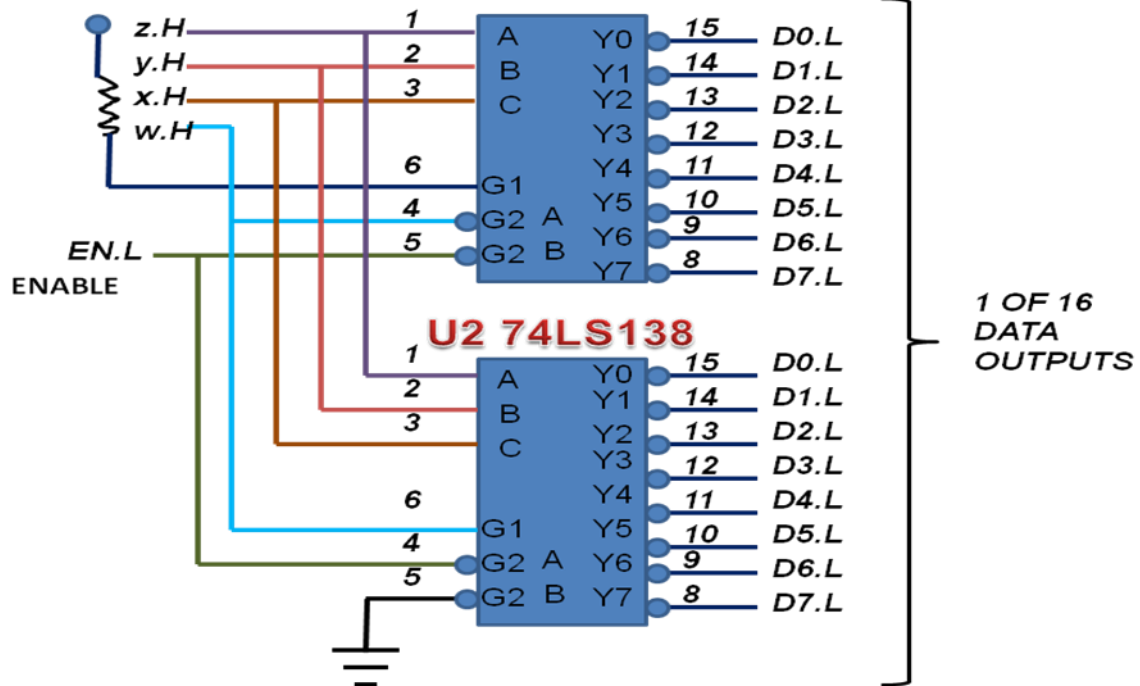
INPUTS			OUTPUTS			
ENABLE		SELECT				
G'	B	A	Y0	Y1	Y2	Y3
H	X	X	H	H	H	H
L	L	L	L	H	H	H

L	L	H	H	L	H	H
L	H	L	H	H	L	H
L	H	H	H	H	H	L

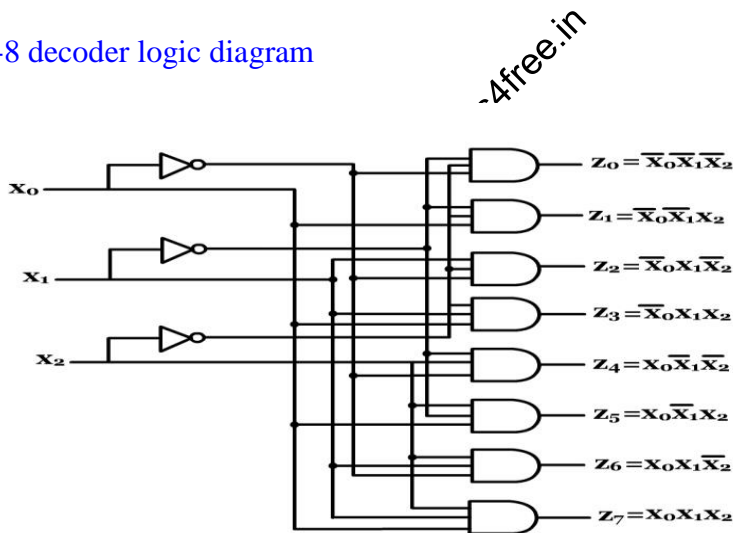


**U1 74LS138**





3-to-8 decoder logic diagram



A 3-to-8 decoder Logic diagram

Inputs			Outputs							
$x_2$	$x_1$	$x_0$	$z_0$	$z_1$	$z_2$	$z_3$	$z_4$	$z_5$	$z_6$	$z_7$
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Truth table.

In this realization shown above the three inputs are assigned  $x_0, x_1,$  and  $x_2$ , and the eight outputs are  $Z_0$  to  $Z_7$ .

Function specific decoders also exist which have less than  $2^n$  outputs . examples are 8421 code decoder also called BCD to decimal decoder. Decoders that drive seven segment displays also exist.

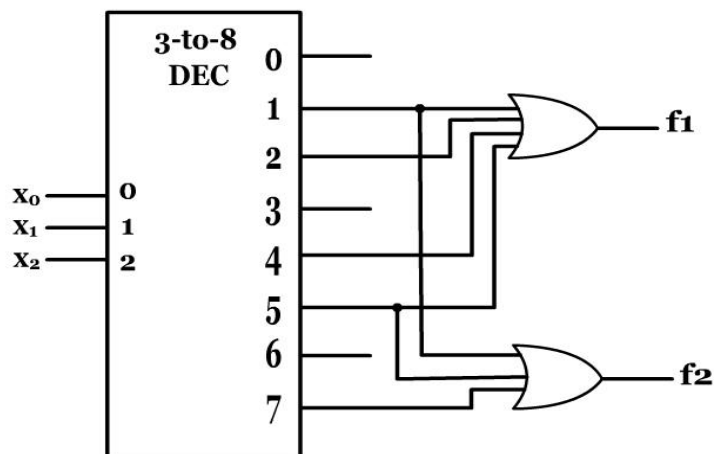
### Realization of boolean expression using Decoder and OR gate

We see from the above truth table that the output expressions correspond to a single minterm. Hence a  $n$  –to  $2^n$  decoder is a minterm generator. Thus by using OR gates in conjunction with a  $n$  –to  $2^n$  decoder boolean function realization is possible.

Ex: to realize the Boolean functions given below using decoders...

- $F_1 = \sum m(1, 2, 4, 5)$

- $F_2 = \sum m(1, 5, 7)$



### Realisation of boolean expressions

Priority encoder  
8-3 line priority encoder

In priority encoder a priority scheme is assigned to the input lines so that whenever more than one input line is asserted at any time, the output is determined by the input line having the highest priority.

The Valid bit is used to indicate that atleast one input line is asserted. This is done to distinguish the situation that no input line is asserted from when the X0 input line is asserted , since in both cases  $Z_1Z_2Z_3 = 000$ .

Inputs								Outputs			
X <sub>0</sub>	X <sub>1</sub>	X <sub>2</sub>	X <sub>3</sub>	X <sub>4</sub>	X <sub>5</sub>	X <sub>6</sub>	X <sub>7</sub>	Z <sub>2</sub>	Z <sub>1</sub>	Z <sub>0</sub>	Valid
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
X	1	0	0	0	0	0	0	0	0	1	1
X	X	1	0	0	0	0	0	0	1	0	1
X	X	X	1	0	0	0	0	0	1	1	1
X	X	X	X	1	0	0	0	1	0	0	1
X	X	X	X	X	1	0	0	1	0	1	1
X	X	X	X	X	X	1	0	1	1	0	1
X	X	X	X	X	X	X	1	1	1	1	1

Condensed truth table for an 8-to-3 line  
priority encoder

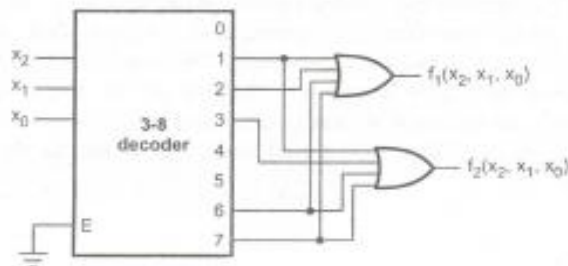
**Recommended question and answer –unit-3**

**Jan 2009**

**Q.3 a)** Realize the following functions expressed in maxterm canonical form in two possible ways using 3-8 line and decoder :

$$f_1(x_2, x_1, x_0) = \pi M (1, 2, 6, 7), \quad f_2(x_2, x_1, x_0) = \pi M (1, 3, 6, 7) \quad (10)$$

Ans. :



b) What are the problems associated with the basic encoder? Explain, how can these problems be overcome by priority encoder, considering 8 input lines. (10)

Ans. : The basic encoder has ambiguity that when all inputs are 0 then outputs are 0s. The zero output can also be generated when  $D_0 = 1$ . This ambiguity can be resolved by providing an additional output that specifies the valid condition.

Inputs								Outputs		
$D_7$	$D_6$	$D_5$	$D_4$	$D_3$	$D_2$	$D_1$	$D_0$	$Y_2$	$Y_1$	$Y_0$
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1
0	0	0	0	0	0	0	0	0	0	0

A priority encoder is an encoder circuit that indicates the priority function. In priority encoder, if two or more inputs are equal to 1 at the same time, the inputs having the highest priority will take precedence. Also, the output V (valid output indicator) indicates one or more of the inputs are equal to 1. If all inputs are '0', V is equal to 0 and other two outputs of the circuit are not used.

**Jan 2008**

Final expression

$$Y = \bar{A} B \bar{C} + B \bar{C} D + \bar{B} C + A \bar{B} + A \bar{D}$$

Q.3 a) Design a combinational circuit that will multiply two two-bit binary values.

Ans. :

A		B		Output		
A <sub>1</sub>	A <sub>0</sub>	B <sub>1</sub>	B <sub>0</sub>	A = B	A > B	A < B
0	0	0	0	1	0	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	0	1	0
0	1	0	1	1	0	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	0	1	0
1	0	0	1	0	1	0
1	0	1	0	1	0	0
1	0	1	1	0	1	0
1	1	0	0	0	1	0
1	1	0	1	0	1	0
1	1	1	0	0	1	0
1	1	1	1	1	0	0

$$y = \bar{A}_0 \bar{A}_1 \bar{B}_0 \bar{B}_1 + B_0 B_1 \bar{A}_1 A_0 + B_0 B_1 A_1 A_0 + B_1 \bar{B}_0 A_1 \bar{A}_0$$

A = B

		A <sub>1</sub> A <sub>0</sub>			
		00	01	11	10
B <sub>1</sub> B <sub>0</sub>	00	1	0	0	0
	01	0	1	0	0
	11	0	0	1	0
	10	0	0	0	1

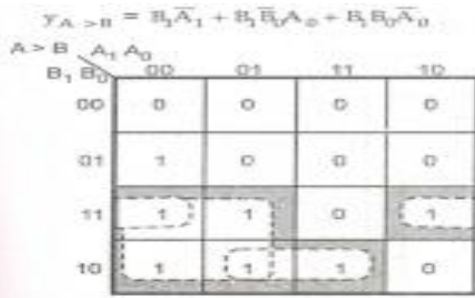


Fig. 7

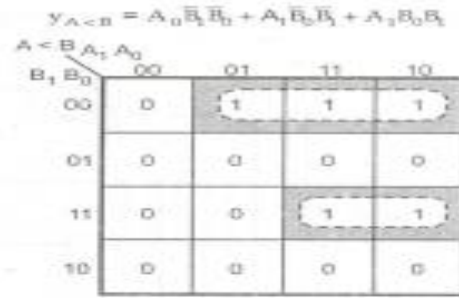
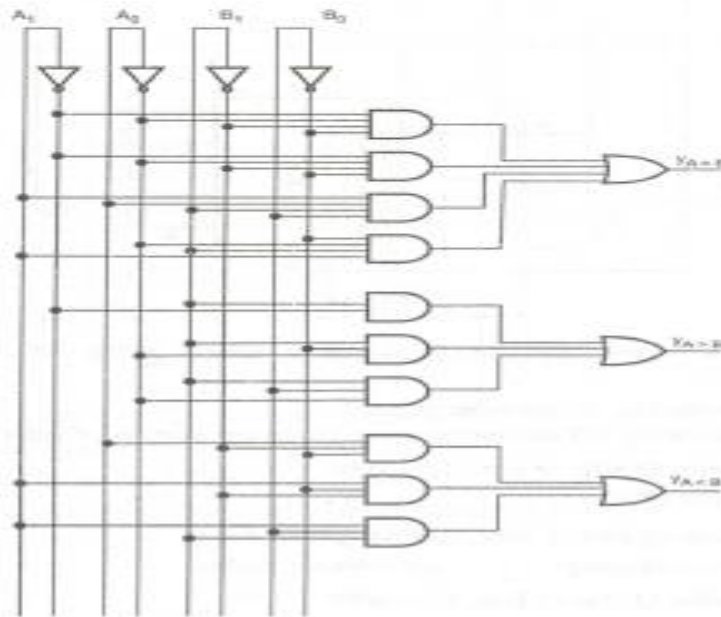


Fig. 8

Design



b) Design a 4 to 16 decoder using two 3 to 8 decoder (74LS138).

Ans. :

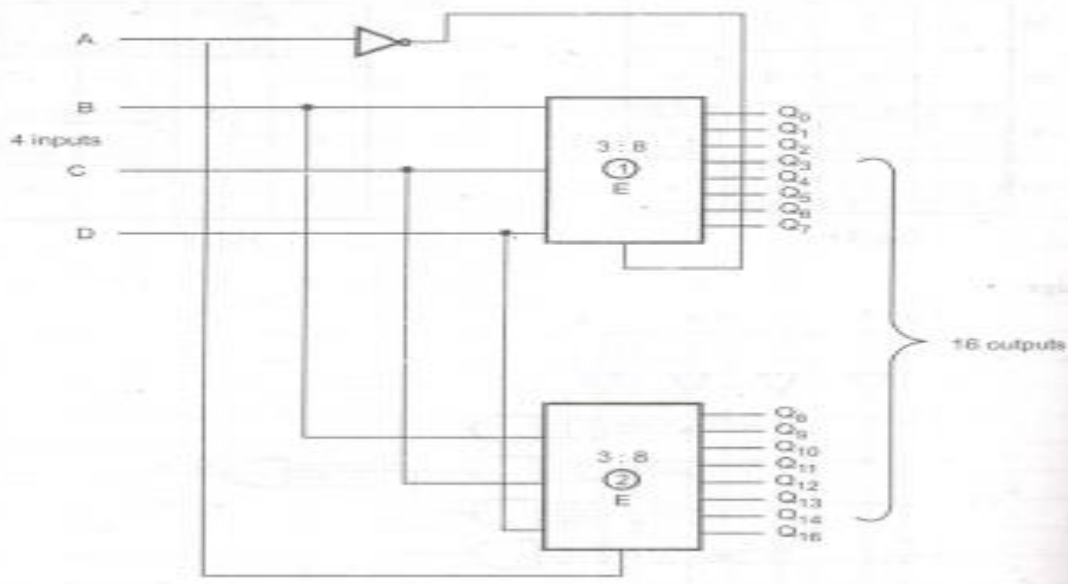


Fig. 10

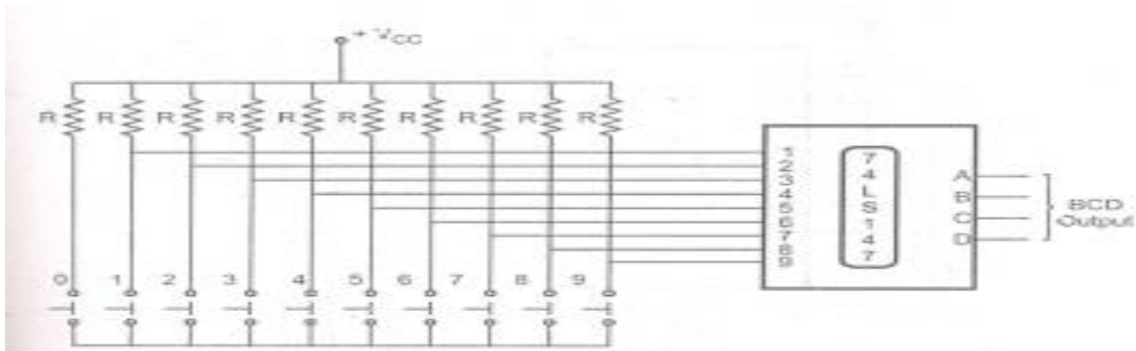


Fig. 11 Design of keypad interface to a digital system using BCD encoder IC 74147

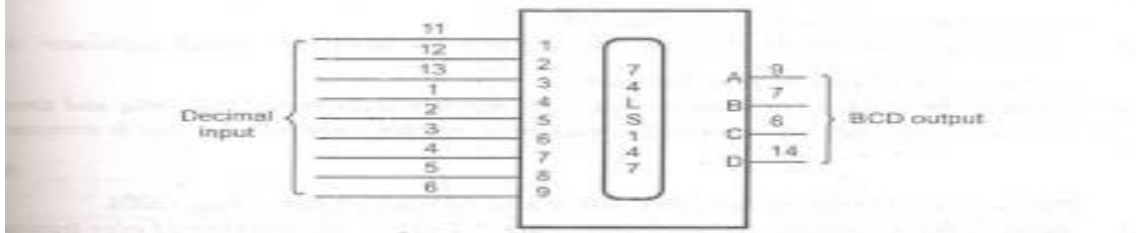


Fig. 12 Pin configuration of 74LS147

Aug 2009

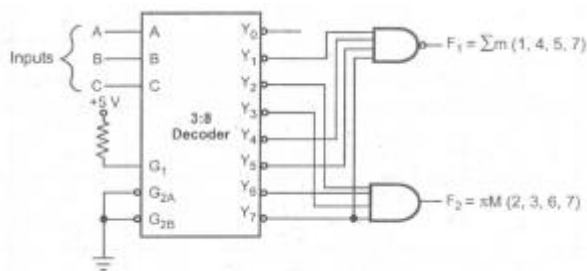
notes4free.in

Q.3 a) Implement following multiple output function using 74LS138 and extend gates.

$$f_1(A, B, C) = \sum m(1, 4, 5, 7)$$

$$f_2(A, B, C) = \prod M(2, 3, 6, 7)$$

Ans. : In this example, we use 74LS138, 3 8 decoder to implement multiple output function. The outputs of 74LS138 are active low, therefore, SOP function (function F1) can be implemented using NAND gate and POS function (function F2) can be implemented using AND gate, as shown in Fig. 6.

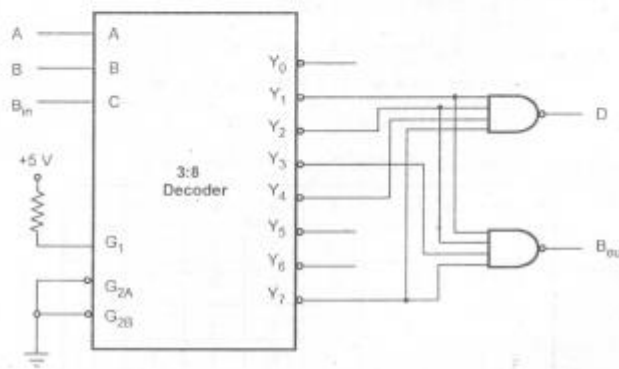


b) Implement full subtractor using decoder and write a truth table.

Ans. : The truth table for full subtractor is as shown in Table 1.

Inputs			Outputs	
A	B	B <sub>in</sub>	D	B <sub>out</sub>
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Table 1 Truth table for full subtractor



nc

c) Write a note on encoders.

(6)

Ans. : An encoder is a digital circuit that performs the inverse operation of a decoder. An encoder has  $2^n$  (or fewer) input lines and  $n$  output lines: In encoder the

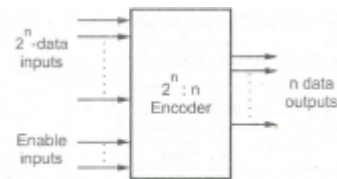


Fig. 8 General structure of encoder

output lines generate the binary code corresponding to the input value. The Fig. 8 shows the general structure of the encoder circuit. As shown in the Fig. 8, the decoded information is presented as  $2^n$  inputs producing  $n$  possible outputs.



Aug-2007

Q.5 a) What is a magnitude comparator? Design a 2 bit digital comparator by writing truth table, the relevant expression, and logic diagram.

Sol. : The truth table for 2-bit comparator is given in Table 1.

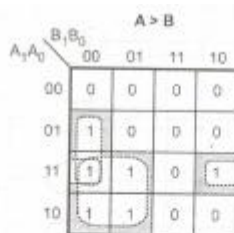
Inputs				Outputs		
A <sub>1</sub>	A <sub>0</sub>	B <sub>1</sub>	B <sub>0</sub>	A > B	A = B	A < B
0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	1	0	0
0	1	0	1	0	1	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0

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

Table 1

K-map simplification

$$\begin{aligned}
 (A = B) &= \bar{A}_1 \bar{A}_0 \bar{B}_1 \bar{B}_0 + \bar{A}_1 A_0 \bar{B}_1 B_0 \\
 &\quad + A_1 A_0 B_1 B_0 + A_1 \bar{A}_0 B_1 \bar{B}_0 \\
 &= \bar{A}_1 \bar{B}_1 (\bar{A}_0 \bar{B}_0 + A_0 B_0) \\
 &\quad + A_1 B_1 (A_0 B_0 + \bar{A}_0 \bar{B}_0) \\
 &= (A_0 \odot B_0) (A_1 \odot B_1) \\
 (A < B) &= \bar{A}_1 \bar{A}_0 B_0 + \bar{A}_0 B_1 B_0 + \bar{A}_1 B_1
 \end{aligned}$$



$$A > B = A_0 \bar{B}_1 \bar{B}_0 + A_1 \bar{B}_1 + A_1 A_0 \bar{B}_0$$

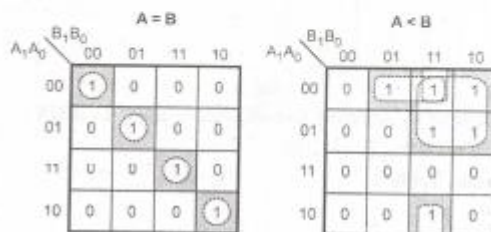


Fig. 8

Logic Diagram

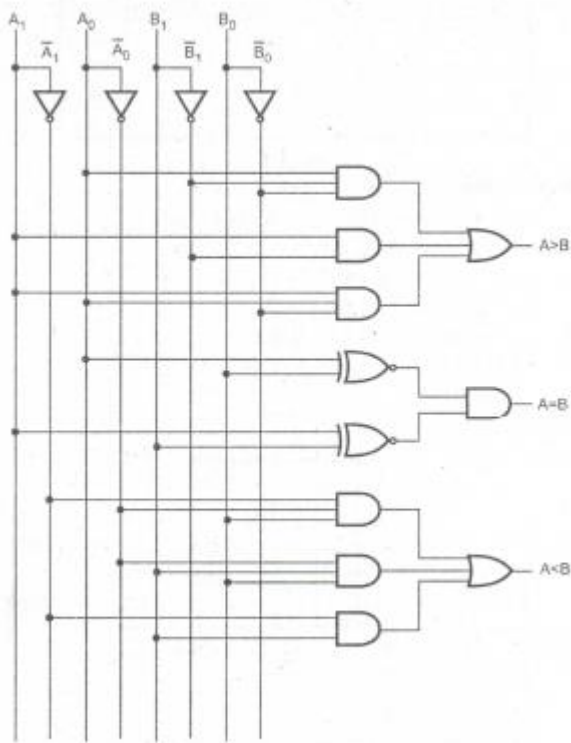


Fig. 9

notes4fr

## **Analysis and design of combinational logic - II:**

Digital multiplexers-Using multiplexers as Boolean function generators. Adders and subtractors - Cascading full adders, Look ahead carry, Binary comparators

### **Recommended readings:**

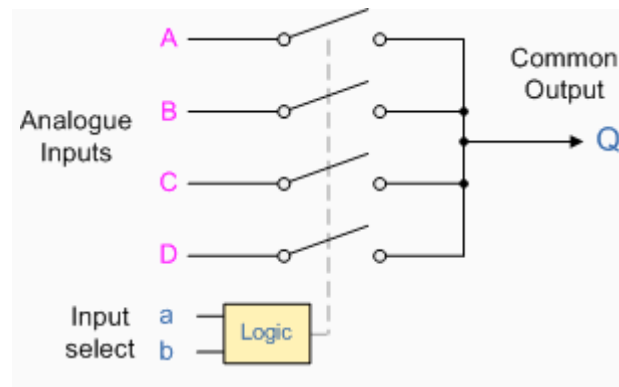
1. John M Yarbrough, "Digital Logic Applications and Design", Thomson Learning, 2001.

Units- 4.5, 4.6 - 4.6.1, 4.6.2, 4.7

### **The Multiplexer**

The Multiplexer which sometimes are simply called "Mux" or "Muxes", are devices that act like a very fast acting rotary switch. They connect multiple input lines 2, 4, 8, 16 etc one at a time to a common output line and are used as one method of reducing the number of logic gates required in a circuit. Multiplexers are individual Analogue Switches as opposed to the "mechanical" types such as normal conventional switches and relays. They are usually made from MOSFETs devices encased in a single package and are controlled using standard logic gates. An example of a Multiplexer is shown below.

#### 4-to-1 Channel Multiplexer



Addressing		Input Selected
b	a	
0	0	A
0	1	B
1	0	C
1	1	D

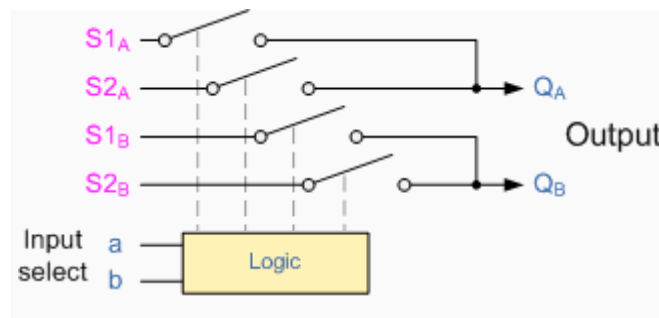
The Boolean expression for this 4 to 1 **Multiplexer** is given as:

$$Q = a\bar{b}A + a\bar{b}B + ab\bar{C} + abD$$

In this example at any instant in time only one of the four analogue switches is closed, connecting only one of the input lines A to D to the single output at Q. As to which switch is closed depends upon the addressing input code on lines "a" and "b", so for this example to select input B to the output at Q, the binary input address would need to be "a" = logic "1" and "b" = logic "0". Adding more control address lines will allow the multiplexer to control more inputs. Multiplexers can also be used to switch either analogue, digital or video signals, with the switching current in analogue circuits limited to below 10mA to 20mA per channel in order to reduce heat dissipation.

Multiplexers are not limited to just switching a number of different input lines or channels to one common single output. There are also types that can switch their inputs to multiple outputs and have arrangements or 4 to 2, 8 to 3 or even 16 to 4 etc configurations and an example of a simple Dual channel 4 input multiplexer (4 to 2) is given below:

**4-to-2 Channel Multiplexer**

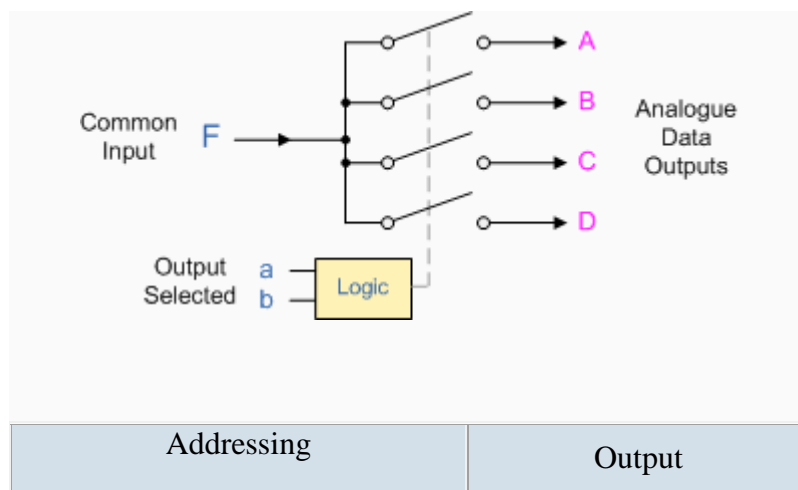


Here in this example the 4 input channels are switched to 2 individual output lines but larger arrangements are also possible. This simple 4 to 2 configuration could be used for example, to switch audio signals for stereo pre-amplifiers or mixers.

The De-multiplexer

**De-multiplexers** or "De-muxes", are the exact opposite of the **Multiplexers** we saw in the previous tutorial in that they have one single input data line and then switch it to any one of their individual multiple output lines one at a time. **The De-multiplexer** converts the serial data signal at the input to a parallel data at its output lines as shown below.

**1-to-4 Channel De-multiplexer**



b	a	Selected
0	0	A
0	1	B
1	0	C
1	1	D

The Boolean expression for this De-multiplexer is given as:

$$F = abA + abB + abC + abD$$

The function of the De-multiplexer is to switch one common data input line to any one of the 4 output data lines A to D in our example above. As with the multiplexer the individual solid state switches are selected by the binary input address code on the output select pins "a" and "b" and by adding more address line inputs it is possible to switch more outputs giving a 1-to-2<sup>n</sup> data lines output. Some standard De-multiplexer IC's also have an "enable output" input pin which disables or prevents the input from being passed to the selected output. Also some have latches built into their outputs to maintain the output logic level after the address inputs have been changed. However, in standard decoder type circuits the address input will determine which single data output will have the same value as the data input with all other data outputs having the value of logic "0".

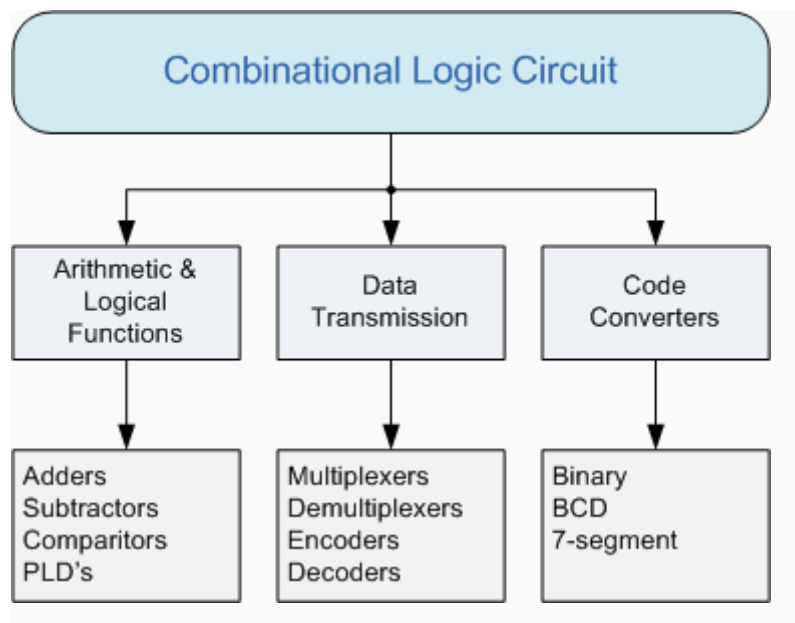
Standard De-multiplexer IC packages available are the TTL 74LS138 1 to 8-output De-multiplexer, the TTL 74LS139 Dual 1 to 4-output De-multiplexer or the CMOS CD4514 1 to 16-output De-multiplexer. Another type of De-multiplexer is the 24-pin, 74LS154 which is a 4-bit to 16-line De-multiplexer/decoder. Here the output positions are selected using the 4-bit binary coded input.

### Combination Logic

Unlike Sequential Logic circuits whose outputs are dependant on both the present input and their previous output state giving them some form of Memory, the outputs of Combinational Logic circuits are only determined by their current input state as they have no feedback, and any changes to the signals being applied to their inputs will immediately have an effect at the output. In other words, in a Combination Logic circuit, if the input condition changes state so too does the output as combinational circuits have No Memory.

Combination Logic circuits are made up from basic logic AND, OR or NOT gates that are "combined" or connected together to produce more complicated switching circuits.

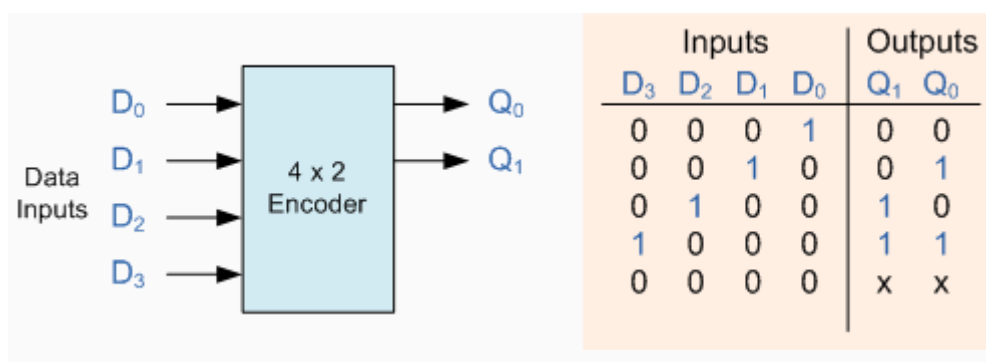
As combination logic circuits are made up from individual logic gates they can also be considered as "decision making circuits" and combinational logic is about combining logic gates together to process two or more signals in order to produce at least one output signal according to the logical function of each logic gate. Common combinational circuits made up from individual logic gates include Multiplexers, Decoders and De-multiplexers, Full and Half Adders etc.

Classification of Combinational Logic

One of the most common uses of combination logic is in Multiplexer and De-multiplexer type circuits. Here, multiple inputs or outputs are connected to a common signal line and logic gates are used to decode an address to select a single data input or output switch. A multiplexer consist of two separate components, a logic decoder and some solid state switches, but before we can discuss multiplexers, decoders and de-multiplexers in more detail we first need to understand how these devices use these "solid state switches" in their design.

The Encoder

Unlike a multiplexer that selects one individual data input line and then sends that data to a single output line or switch, an Encoder takes all the data inputs one at a time and converts them to a single encoded output. Then, it is a multi-input data line, combinational logic circuit that converts the logic level "1" data at its inputs to an equivalent binary code at its output. Generally encoders produce outputs of 2-bit, 3-bit or 4-bit codes depending upon the number of data input lines and a "n-bit" encoder has  $2^n$  input lines with common types that include 4-to-2, 8-to-3 and 16-to-4 line configurations. Encoders are available to encode either a decimal or hexadecimal input pattern to typically a binary or B.C.D. output code.

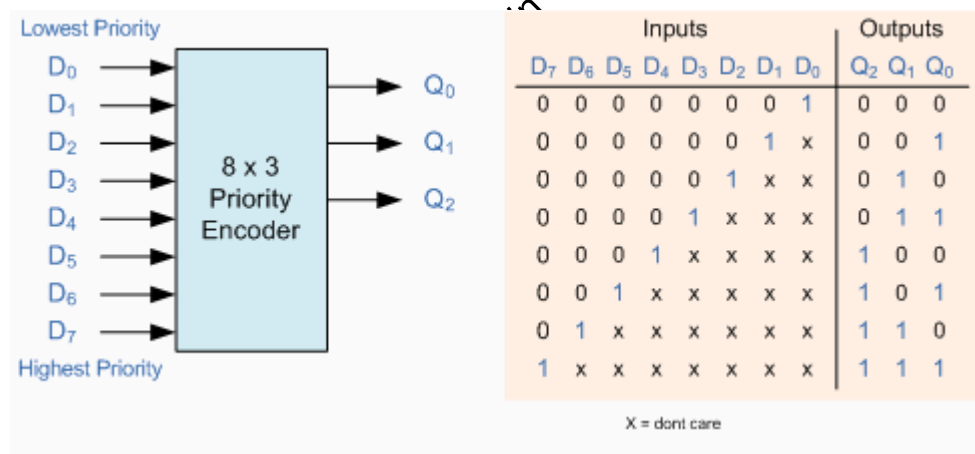
4-to-2 Bit Encoder

One of the main disadvantages of standard encoders is that they can generate the wrong output code when there is more than one input present at logic level "1". For example, if we make inputs D1 and D2 HIGH at logic "1" at the same time, the resulting output is neither at "01" or at "10" but will be at "11" which is an output code that is different to the actual input present. One simple way to overcome this problem is to "Prioritize" the level of each input pin and if there was more than one input at logic level "1" the actual output code would only correspond to the input with the highest designated priority. Then this type of encoder are known as Priority Encoders or P-encoder.

### Priority Encoders

**Priority Encoders** come in many forms and an example of an 8-input Priority Encoder along with its truth table is as shown below.

#### 8-to-3 Bit Priority Encoder



Priority encoders are available in standard IC form and the TTL 74LS148 is an 8 to 3 bit priority encoder which has eight active LOW (logic "0") inputs and provides a 3-bit code of the highest ranked input at its output. Priority encoders output the highest order input first for example, if input lines "D2", "D3" and "D5" are applied simultaneously the output code would be for input "D5" ("101") as this has the highest order out of the 3 inputs. Once input "D5" had been removed the next highest output code would be for input "D3" ("011"), and so on.

### Encoder Applications

#### Keyboard Encoders

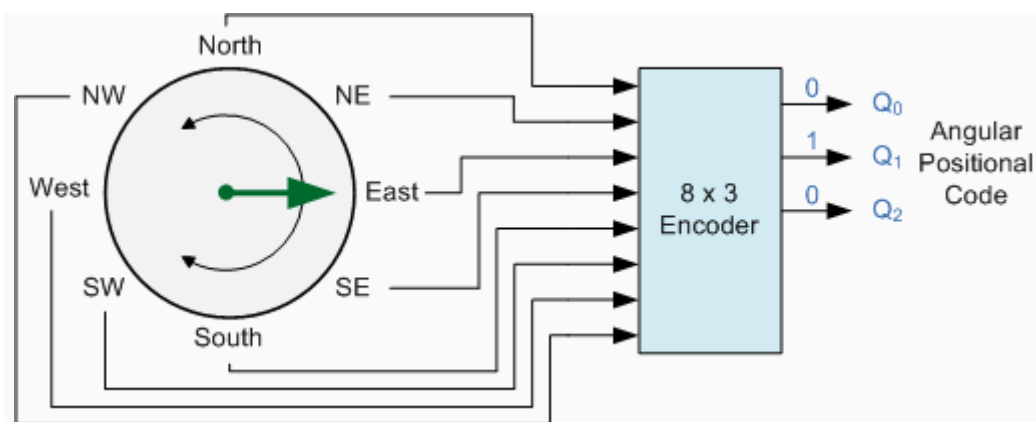
Priority encoders can be used to reduce the number of wires needed in circuits or applications that has multiple inputs. For example, assume that a microcomputer needs to read the 104 keys of a standard QWERTY keyboard where only one key would be pressed or HIGH at any



one time. One way would be to connect all 104 wires from the keys directly to the computer but this would be impractical for a small home PC, but another better way would be to use an encoder. The 104 individual buttons or keys could be encoded into a standard ASCII code of only 7-bits (0 to 127 decimal) to represent each key or character and then inputted as a much smaller 7-bit B.C.D code directly to the computer. Keypad encoders such as the 74C923 20-key encoder are available.

### Positional Encoders

Another more common application is in magnetic positional control as used on ships or robots etc. Here the angular or rotary position of a compass is converted into a digital code by an encoder and inputted to the systems computer to provide navigational data and an example of a simple 8 position to 3-bit output compass encoder is shown below.



Compass Direction	Binary Output		
	Q <sub>0</sub>	Q <sub>1</sub>	Q <sub>2</sub>
North	0	0	0
North-East	0	0	1
East	0	1	0
South-East	0	1	1
South	1	0	0
South-West	1	0	1
West	1	1	0
North-West	1	1	1

### Interrupt Requests

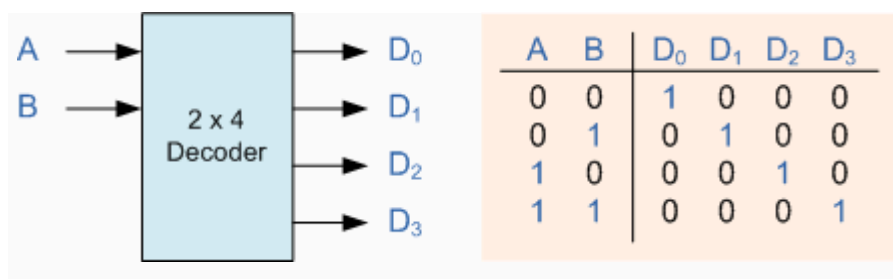
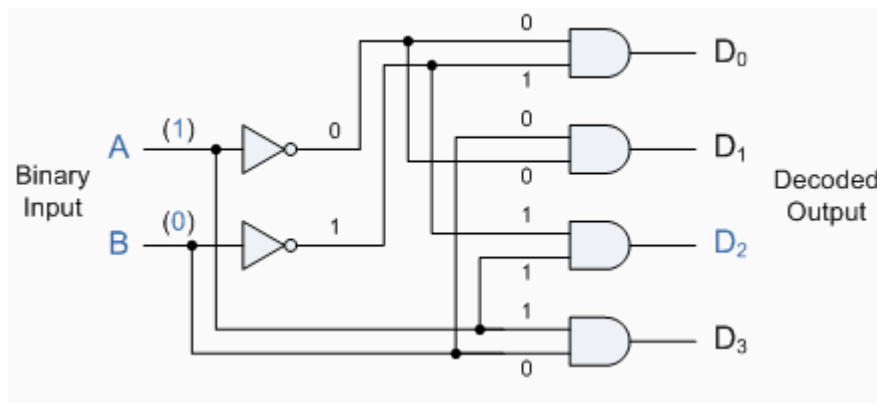
Other applications for Priority Encoders may include detecting interrupts in microprocessor applications. Here the microprocessor uses interrupts to allow peripheral devices such as the disk drive, scanner, mouse, or printer etc, to communicate with it, but the microprocessor can only "talk" to one peripheral device at a time. The processor uses "Interrupt Requests" or "IRQ" signals to assign priority to the devices to ensure that the most important peripheral device is serviced first. The order of importance of the devices will depend upon their connection to the priority encoder.

Because implementing such a system using priority encoders such as the standard 74LS148 priority encoder IC involves additional logic circuits, purpose built integrated circuits such as the 8259 Programmable Priority Interrupt Controller is available.

IRQ Number	Typical Use	Description
IRQ 0	System timer	Internal System Timer.
IRQ 1	Keyboard	Keyboard Controller.
IRQ 3	COM2 & COM4	Second and Fourth Serial Port.
IRQ 4	COM1 & COM3	First and Third Serial Port.
IRQ 5	Sound	Sound Card.
IRQ 6	Floppy disk	Floppy Disk Controller.
IRQ 7	Parallel port	Parallel Printer.
IRQ 12	Mouse	PS/2 Mouse.
IRQ 14	Primary IDE	Primary Hard Disk Controller.
IRQ 15	Secondary IDE	Secondary Hard Disk Controller.

### [Binary Decoders](#)

A Decoder is the exact opposite to that of an "Encoder" we looked at in the last tutorial. It is basically, a combinational type logic circuit that converts the binary code data at its input into one of a number of different output lines, one at a time producing an equivalent decimal code at its output. Binary Decoders have inputs of 2-bit, 3-bit or 4-bit codes depending upon the number of data input lines, and a "n-bit" decoder has  $2^n$  output lines. Typical combinations of decoders include, 2-to-4, 3-to-8 and 4-to-16 line configurations. Binary Decoders are available to "decode" either a Binary or BCD input pattern to typically a Decimal output code.

**A 2-to-4 Binary Decoders.**

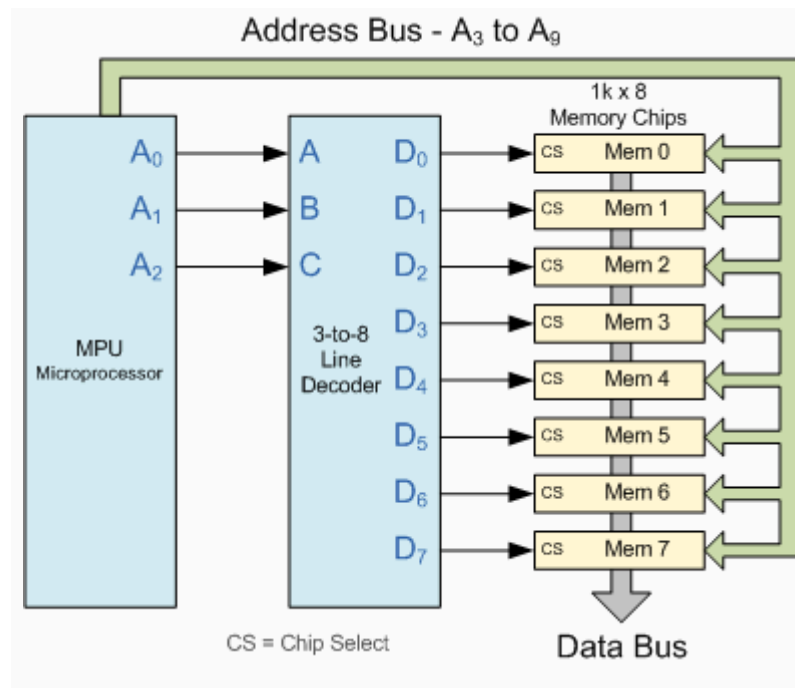
In this simple example of a 2-to-4 line binary decoder, the binary inputs A and B determine which output line from D<sub>0</sub> to D<sub>3</sub> is "HIGH" at logic level "1" while the remaining outputs are held "LOW" at logic "0". Therefore, whichever output line is "HIGH" identifies the binary code present at the input, in other words it "de-codes" the binary input and these types of binary decoders are commonly used as Address Decoders in microprocessor memory applications.

**Memory Address Decoder.**

In modern microprocessor systems the amount of memory required can be quite high and is generally more than one single memory chip alone. One method of overcoming this problem is to connect lots of individual memory chips together and to read the data on a common "Data Bus". In order to prevent the data being "read" from each memory chip at the same time, each memory chip is selected individually one at a time and this process is known as Address Decoding.

Each memory chip has an input called Chip Select or CS which is used by the MPU to select the appropriate memory chip and a logic "1" on this input selects the device and a logic "0" on the input de-selects it. By selecting or de-selecting each chip, allows us to select the correct memory device for a particular address and when we specify a particular memory address, the corresponding memory location exists ONLY in one of the chips.

For example, Lets assume we have a very simple microprocessor system with only 1Kb of RAM memory and 10 address lines. The memory consists of 128x8-bit (128x8 = 1024 bytes) devices and for 1Kb we will need 8 individual memory devices but in order to select the correct memory chip we will also require a 3-to-8 line binary decoder as shown below.

**Memory Address Decoding.**

The binary decoder requires 3 address lines, (A<sub>0</sub> to A<sub>2</sub>) to select each one of the 8 chips (the lower part of the address), while the remaining 7 address lines (A<sub>3</sub> to A<sub>9</sub>) select the correct memory location on that chip (the upper part of the address). Having selected a memory location using the address bus, the information at the particular internal memory location is sent to the "Data Bus" for use by the microprocessor. This is of course a simple example but the principals remain the same for all types of memory chips or modules.

**Binary Decoders** are very useful devices for converting one digital format to another, such as binary or BCD type data into decimal or octal etc and commonly available decoder IC's are the TTL 74LS138 3-to-8 line binary decoder or the 74ALS154 4-to-16 line decoder. They are also very useful for interfacing to 7-segment displays such as the TTL 74LS47 which we will look at in the next tutorial.

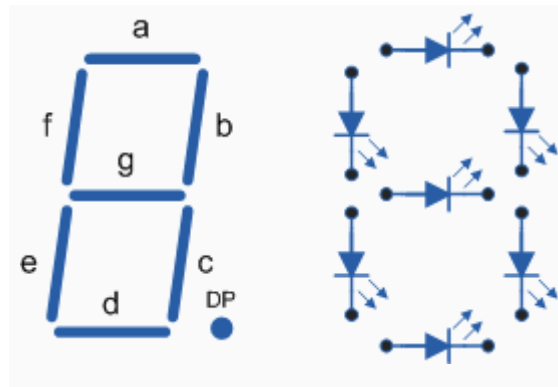
**Display Decoders**

A Decoder IC, is a device which converts one digital format into another and the most commonly used device for doing this is the BCD (Binary Coded Decimal) to 7-Segment Display Decoder. 7-segment LED (Light Emitting Diode) or LCD (Liquid Crystal) Displays, provide a very convenient way of displaying information or digital data in the form of Numbers, Letters or even Alpha-numerical characters and they consist of 7 individual LEDs (the segments), within one single display package. In order to produce the required numbers or characters from 0 to 9 and A to F respectively, on the display the correct combination of LED segments need to be illuminated and Display Decoders do just that. A standard 7-segment LED or LCD display generally has 8 input connections, one for each LED segment and one that acts as a common terminal or connection for all the internal segments. Some single displays have an additional input pin for the decimal point in their lower right or left hand corner.

There are two important types of 7-segment LED digital display.

- The Common Cathode Display (CCD) - In the common cathode display, all the cathode connections of the LEDs are joined together to logic "0" and the individual segments are illuminated by application of a "HIGH", logic "1" signal to the individual Anode terminals.
- The Common Anode Display (CAD) - In the common anode display, all the anode connections of the LEDs are joined together to logic "1" and the individual segments are illuminated by connecting the individual Cathode terminals to a "LOW", logic "0" signal.

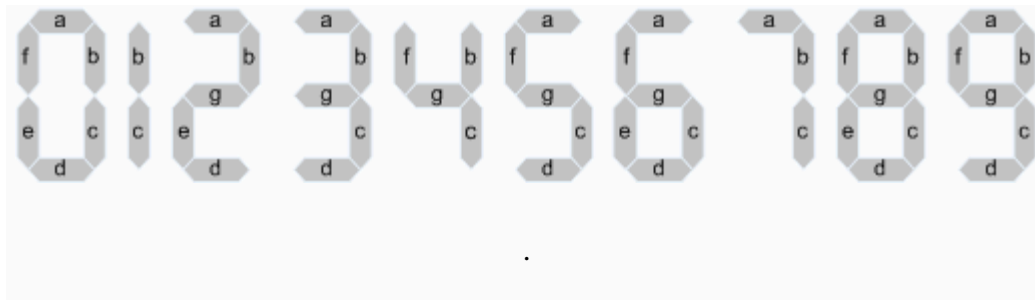
**7-Segment Display Format**



**Truth Table for a 7-segment display**

Individual Segments							Display
a	b	c	d	e	f	g	
x	x	x	x	x	x		0
	x	x					1
x	x		x	x		x	2
x	x	x	x			x	3
	x	x			x	x	4
x		x	x		x	x	5
x		x	x	x	x	x	6
x	x	x					7

Individual Segments							Display
a	b	c	d	e	f	g	
x	x	x	x	x	x	x	8
x	x	x			x	x	9
x	x	x		x	x	x	A
		x	x	x	x	x	b
x			x	x	x		C
	x	x	x	x		x	d
x			x	x	x	x	E
x				x	x	x	F



It can be seen that to display any single digit number from 0 to 9 or letter from A to F, we would need 7 separate segment connections plus one additional connection for the LED's "common" connection. Also as the segments are basically a standard light emitting diode, the driving circuit would need to produce up to 20mA of current to illuminate each individual segment and to display the number 8, all 7 segments would need to be lit resulting a total current of nearly 140mA, (8 x 20mA). Obviously, the use of so many connections and power consumption is impractical for some electronic or microprocessor based circuits and so in order to reduce the number of signal lines required to drive just one single display, display decoders such as the BCD to 7-Segment Display Decoder and Driver IC's are used instead.

### Binary Coded Decimal

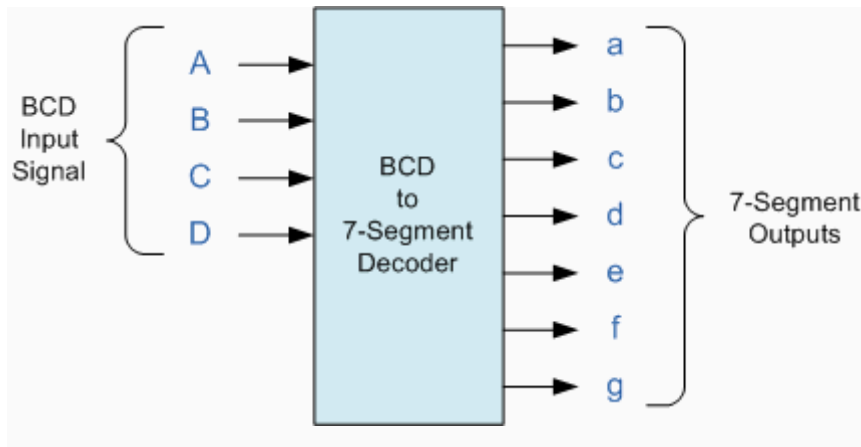
Binary Coded Decimal (BCD or "8421" BCD) numbers are made up using just 4 data bits (a nibble or half a byte) similar to the Hexadecimal numbers we saw in the binary tutorial, but unlike hexadecimal numbers that range in full from 0 through to F, BCD numbers only range from 0 to 9, with the binary number patterns of 1010 through to 1111 (A to F) being invalid inputs for this type of display and so are not used as shown below.

Decimal	Binary Pattern				BCD	Decimal	Binary Pattern				BCD
	8	4	2	1			8	4	2	1	
0	0	0	0	0	0	8	1	0	0	0	8
1	0	0	0	1	1	9	1	0	0	1	9
2	0	0	1	0	2	10	1	0	1	0	Invalid
3	0	0	1	1	3	11	1	0	1	1	Invalid
4	0	1	0	0	4	12	1	1	0	0	Invalid
5	0	1	0	1	5	13	1	1	0	1	Invalid
6	0	1	1	0	6	14	1	1	1	0	Invalid
7	0	1	1	1	7	15	1	1	1	1	Invalid

BCD to 7-Segment Display Decoders

A binary coded decimal (BCD) to 7-segment display decoder such as the TTL 74LS47 or 74LS48, have 4 BCD inputs and 7 output lines, one for each LED segment. This allows a smaller 4-bit binary number (half a byte) to be used to display all the denary numbers from 0 to 9 and by adding two displays together, a full range of numbers from 00 to 99 can be displayed with just a single byte of 8 data bits.

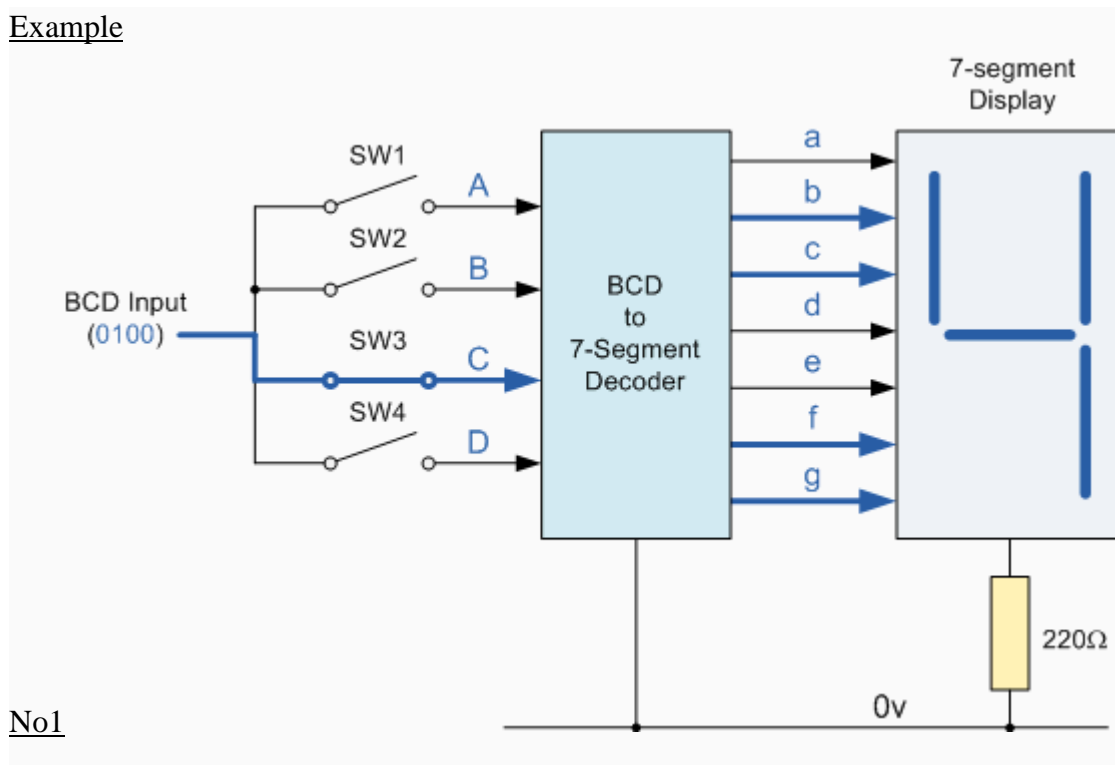
BCD to 7-Segment Decoder



The use of packed BCD allows two BCD digits to be stored within a single byte (8-bits) of data, allowing a single data byte to hold a BCD number in the range of 00 to 99.

An example of the 4-bit BCD input (0100) representing the number 4 is given below.

Example



No1

In practice current limiting resistors of about  $150\Omega$  to  $220\Omega$  would be connected in series between the decoder/driver chip and each LED display segment to limit the maximum current flow. Different display decoders or drivers are available for the different types of display available, e.g. 74LS48 for common-cathode LED types, 74LS47 for common-anode LED types, or the CMOS CD4543 for liquid crystal display (LCD) types.

Liquid crystal displays (LCD's) have one major advantage over similar LED types in that they consume much less power and nowadays, both LCD and LED displays are combined together to form larger Dot-Matrix Alphanumeric type displays which can show letters and characters as well as numbers in standard Red or Tri-colour outputs.

### [The Binary Adder](#)

Another common and very useful combinational logic circuit is that of the Binary Adder circuit. The Binary Adder is made up from standard AND and Ex-OR gates and allow us to "add" single bits of data together to produce two outputs, the SUM ("S") of the addition and a CARRY ("C"). One of the main uses for the Binary Adder is in arithmetic and counting circuits.

Consider the addition of two denary (base 10) numbers below.

123	A	(Augend)
+ 789	B	(Addend)
912	SUM	

nc

Each column is added together starting from the right hand side. As each column is added together a carry is generated if the result is greater or equal to ten, the base number. This carry is then added to the result of the addition of the next column to the left and so on, simple school math's addition. Binary addition is based on similar principals but a carry is only generated when the result in any column is greater or equal to "2", the base number of binary.

### [Binary Addition](#)

Binary Addition follows the same basic rules as for the denary addition above except in binary there are only two digits and the largest digit is "1", so any "SUM" greater than 1 will result in a "CARRY". This carry 1 is passed over to the next column for addition and so on. Consider the single bit addition below.

0	0	1	1
<u>+0</u>	<u>+1</u>	<u>+0</u>	<u>+1</u>
0	1	1	10

The single bits are added together and "0 + 0", "0 + 1", or "1 + 0" results in a sum of "0" or "1" until you get to "1 + 1" then the sum is equal to "2". For a simple 1-bit addition problem like this, the resulting carry bit could be ignored which would result in an output truth table



resembling that of an Ex-OR Gate as seen in the Logic Gates section and whose result is the sum of the two bits but without the carry. An Ex-OR gate only produces an output "1" when either input is at logic "1", but not both. However, all microprocessors and electronic calculators require the carry bit to correctly calculate the equations so we need to rewrite them to include 2 bits of output data as shown below.

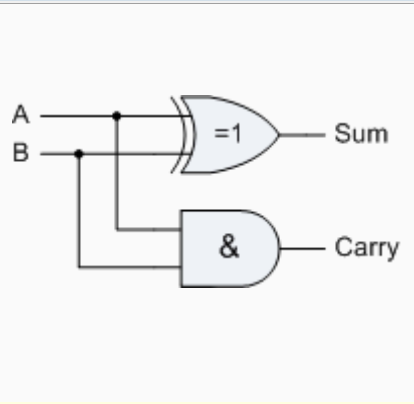
00	00	01	01
<u>+ 00</u>	<u>+ 01</u>	<u>+ 00</u>	<u>+ 01</u>
00	01	01	10

From the above equations we know that an Ex-OR gate will only produce an output "1" when "EITHER" input is at logic "1", so we need an additional output to produce a carry output, "1" when "BOTH" inputs "A" and "B" are at logic "1" and a standard AND Gate fits the bill nicely. By combining the Ex-OR gate with the AND gate results in a simple digital binary adder circuit known commonly as the "Half-Adder" circuit.

### [The Half-Adder Circuit](#)

notes4free.in

### 1-bit Adder with Carry-Out

Symbol	Truth Table			
	A	B	SUM	CARRY
	0	0	0	0
	0	1	1	0
	1	0	1	0
	1	1	0	1
Boolean Expression: Sum = $A \oplus B$ Carry = $A \cdot B$				

From the truth table we can see that the SUM (S) output is the result of the Ex-OR gate and the Carry-out (CO) is the result of the AND gate. One major disadvantage of the Half-Adder circuit when used as a binary adder, is that there is no provision for a "Carry-in" from the previous circuit when adding together multiple data bits. For example, suppose we want to add together two 8-bit bytes of data, any resulting carry bit would need to be able to "ripple" or move across the bit patterns starting from the least significant bit (LSB). As the Half-Adder

has no carry input the resultant added value would be incorrect. One simple way to overcome this problem is to use a "Full-Adder" type binary adder circuit.

### [The Full-Adder Circuit](#)

The main difference between the "Full-Adder" and the previous seen "Half-Adder" is that a Full-Adder has 3-inputs, the two same data inputs "A" and "B" as before plus an additional "Carry-In" (C-in) input as shown below.

### **Full-Adder with Carry-In**

Symbol	Truth Table				
	A	B	C-in	Sum	C-out
	0	0	0	0	0
	0	1	0	1	0
	1	0	0	1	0
	1	1	0	0	1
	0	0	1	1	0
	0	1	1	0	1
	1	0	1	0	1
	1	1	1	1	1
Boolean Expression: $\text{Sum} = A \oplus B \oplus \text{C-in}$					

The Full-Adder circuit above consists of three Ex-OR gates, two AND gates and an OR gate. The truth table for the Full-Adder includes an additional column to take into account the Carry-in input as well as the summed output and Carry-out. 4-bit Full-Adder circuits are available as standard IC packages in the form of the TTL 74LS83 or the 74LS283 which can add together two 4-bit binary numbers and generate a SUM and a CARRY output.

### [The Digital Comparator](#)

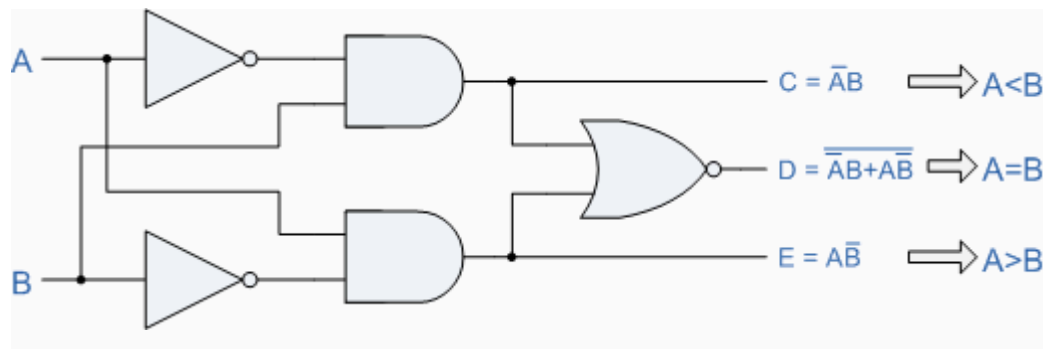
Another common and very useful combinational logic circuit is that of the Digital Comparator circuit. Digital or Binary Comparators are made up from standard AND, NOR and NOT gates that compare the digital signals at their input terminals and produces an output depending upon the condition of the inputs. For example, whether input A is greater than, smaller than or equal to input B etc.

Digital Comparators can compare a variable or unknown number for example A (A1, A2, A3, .... An, etc) against that of a constant or known value such as B (B1, B2, B3, .... Bn, etc) and produce an output depending upon the result. For example, a comparator of 1-bit, (A and B) would produce the following three output conditions.

$$A > B, \quad A = B, \quad A < B$$

This is useful if we want to compare two values and produce an output when the condition is achieved. For example, produce an output from a counter when a certain count number is reached. Consider the simple 1-bit comparator below.

### 1-bit Comparator



Then the operation of a 1-bit digital comparator is given in the following Truth Table.

### Truth Table

Inputs		Outputs		
B	A	A > B	A = B	A < B
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

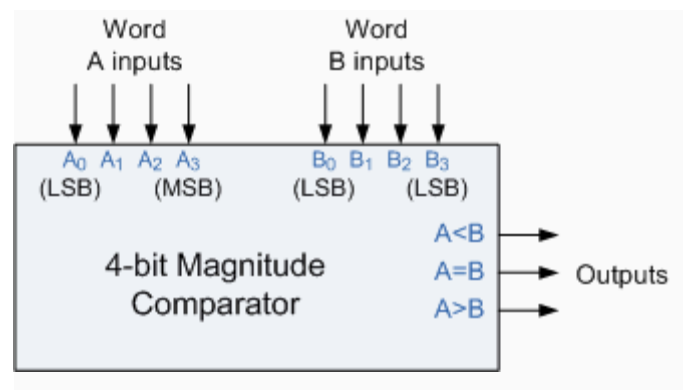
You may notice two distinct features about the comparator from the above truth table. Firstly, the circuit does not distinguish between either two "0" or two "1"s as an output  $A = B$  is produced when they are both equal, either  $A = B = "0"$  or  $A = B = "1"$ . Secondly, the output condition for  $A = B$  resembles that of a commonly available logic gate, the Exclusive-NOR or Ex-NOR gate giving  $Q = A \oplus B$

Digital comparators actually use Exclusive-NOR gates within their design for comparing the respective pairs of bits in each of the two words with single bit comparators cascaded together to produce Multi-bit comparators so that larger words can be compared.

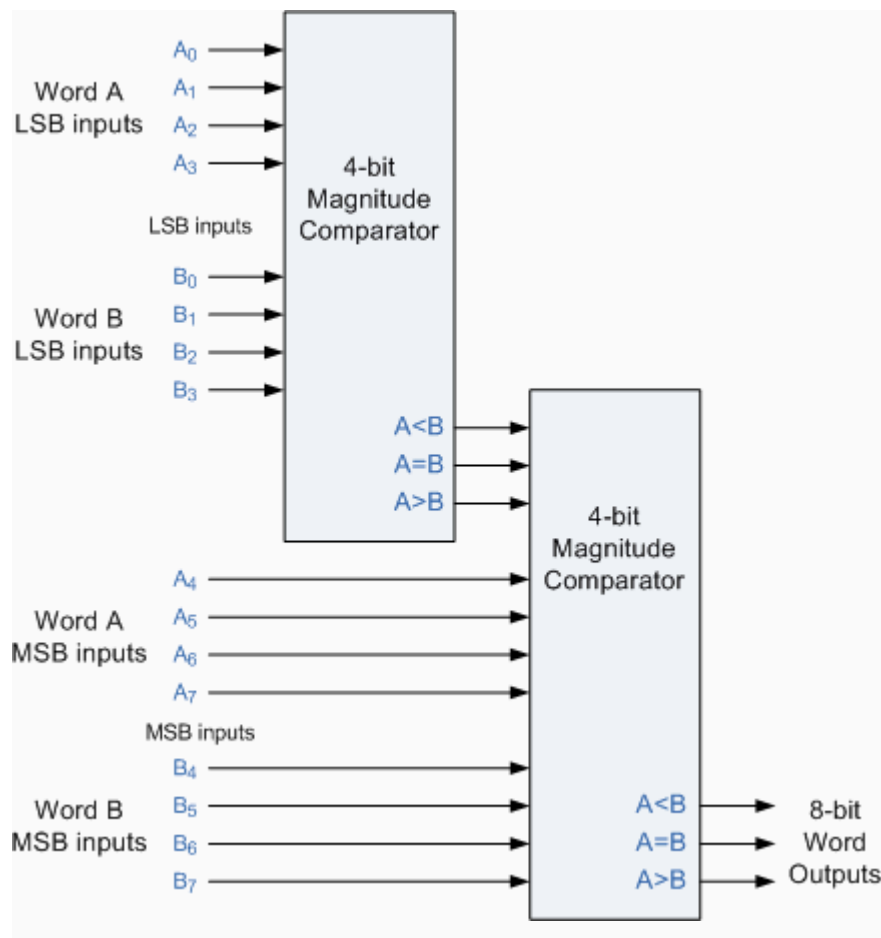
### Magnitude Comparators

As well as comparing individual bits, multi-bit comparators can be constructed to compare whole binary or BCD words to produce an output if one word is larger, equal to or less than the other. A very good example of this is the 4-bit Magnitude Comparator. Here, two 4-bit words ("nibbles") are compared to produce the relevant output with one word connected to inputs A and the other to be compared against connected to input B as shown below.

#### 4-bit Magnitude Comparator



Some commercially available Magnitude Comparators such as the 7485 have additional input terminals that allow more individual comparators to be "cascaded" together to compare words larger than 4-bits with magnitude comparators of "n"-bits being produced. These cascading inputs are connected directly to the corresponding outputs of the previous comparator as shown to compare 8, 16 or even 32-bit words.

**8-bit Word Comparator****Designing a circuit that adds three 4-bit numbers**

Recall that a 4-bit binary adder adds two binary numbers, where each number is of 4 bits.

For adding three 4-bit numbers we have:

**Inputs**

\_ First 4-bit number  $X = X_3X_2X_1X_0$

\_ Second 4-bit number  $Y = Y_3Y_2Y_1Y_0$

\_ Third 4-bit number  $Z = Z_3Z_2Z_1Z_0$

**Outputs**

The summation of  $X$ ,  $Y$ , and  $Z$ . How many output lines are exactly needed will be discussed as we proceed.

To design a circuit using MSI devices that adds three 4-bit numbers, we first have to understand how the addition is done. In this case, the addition will take place in two steps, that is, we will first add the first two numbers, and the resulting sum will be added to the third number, thus giving us the complete addition.

Apparently it seems that we will have to use two 4-bit adders, and probably some extra hardware as well. Let us analyze the steps involved in adding three 4-bit numbers.

### Step 1: Addition of X and Y

A 4-bit adder is required. This addition will result in a sum and a possible carry, as follows:

X<sub>3</sub>X<sub>2</sub>X<sub>1</sub>X<sub>0</sub>

Y<sub>3</sub>Y<sub>2</sub>Y<sub>1</sub>Y<sub>0</sub>

-----

C<sub>4</sub> S<sub>3</sub>S<sub>2</sub>S<sub>1</sub>S<sub>0</sub>

Note that the input carry  $C_{in} = 0$  in this 4-bit adder

### Step 2: Addition of S and Z

This resulting partial sum (i.e. S<sub>3</sub>S<sub>2</sub>S<sub>1</sub>S<sub>0</sub>) will be added to the third 4-bit number Z<sub>3</sub>Z<sub>2</sub>Z<sub>1</sub>Z<sub>0</sub> by using another 4-bit adder as follows, resulting in a final sum and a possible carry:

S<sub>3</sub>S<sub>2</sub>S<sub>1</sub>S<sub>0</sub>

Z<sub>3</sub>Z<sub>2</sub>Z<sub>1</sub>Z<sub>0</sub>

-----

D<sub>4</sub> F<sub>3</sub>F<sub>2</sub>F<sub>1</sub>F<sub>0</sub>

where F<sub>3</sub>F<sub>2</sub>F<sub>1</sub>F<sub>0</sub> represents the final sum of the three inputs X, Y, and Z. Again, in this step, the input carry to this second adder will also be zero.

Notice that in **Step 1**, a carry **C4** was generated in bit position 4, while in **Step 2**, another carry **D4** was generated also in bit position 4. These two carries must be added together to generate the final Sum bits of positions 4 and 5 (**F4** and **F5**).

Adding **C4** and **D4** requires a half adder. Thus, the output from this circuit will be six bits, namely **F5 F4 F3F2F1F0** (See Figure )

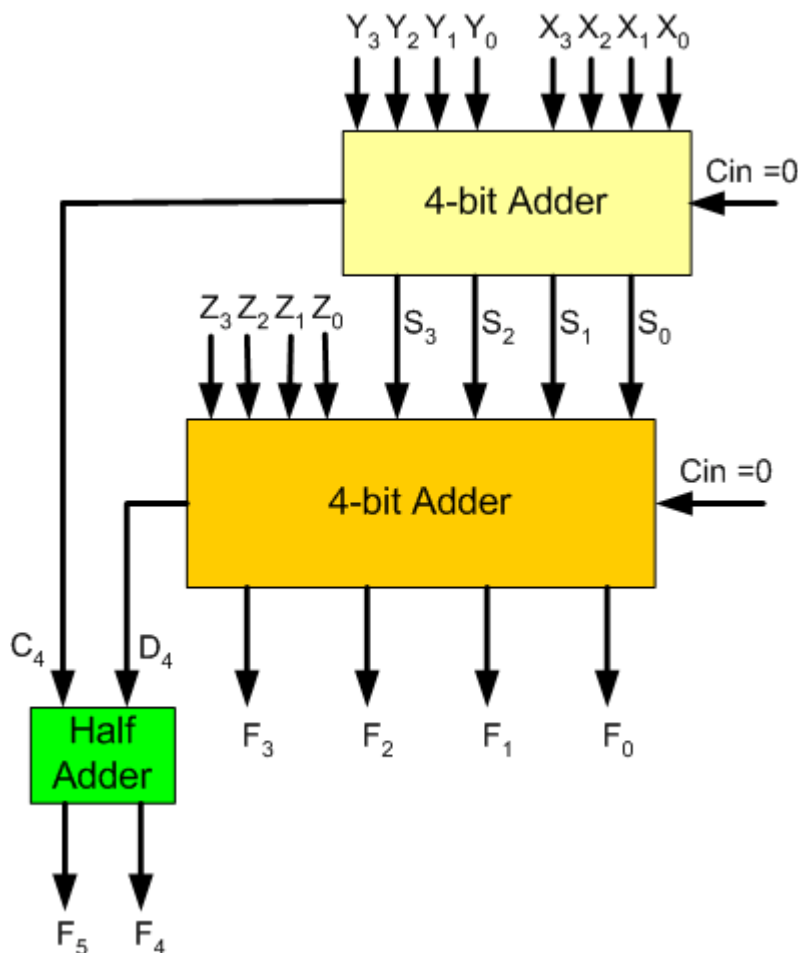


Figure : Circuit for adding three 4-bit numbers

### Design a 4-to-16 Decoder using five 2-to-4 Decoders with enable inputs

We have seen how can we construct a bigger decoder using smaller decoders, by taking the specific example of designing a 3-to-8 decoder using two 2-to-4 decoders. Now we will design a 4-to-16 decoder using five 2-to-4 decoders. There are a total of sixteen possible input combinations, as shown in the table (Figure ). These sixteen combinations can be

divided into four groups, each group containing four combinations. Within each group, A3 and A2 remain constant, while A1 and A0 change their values. Also, in each group, same combination is repeated for A1 and A0

(i.e.00→01→10→11)

A <sub>3</sub>	A <sub>2</sub>	A <sub>1</sub>	A <sub>0</sub>
0	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	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

notes4free.in

Figure : Combinations with 4 variables

Thus we can use a 2-to-4 decoder for each of the groups, giving us a total of four decoders (since we have sixteen outputs; each decoder would give four outputs). To each decoder, A1 and A0 will go as the input. A fifth decoder will be used to select which of the four other decoders should be activated. The inputs to this fifth decoder will be A3 and A2. Each of the four outputs of this decoder will go to each enable of the other four decoders in the “proper order”. This means that line 0 (representing A3A2 = 00) of decoder ‘5’ will go to the enable of decoder ‘1’. Line 1 (representing A3A2 = 01) of decoder ‘5’ will go to the enable of decoder ‘2’ and so on. Thus a combination of A3 and A2 will decide which “group” (decoder) to select, while the combination of A1 and A0 will decide which output line of that particular decoder is to be selected. Moreover, the enable input of decoder ‘5’ will be connected to logic switch, which will provide logic 1 value to activate the decoder.



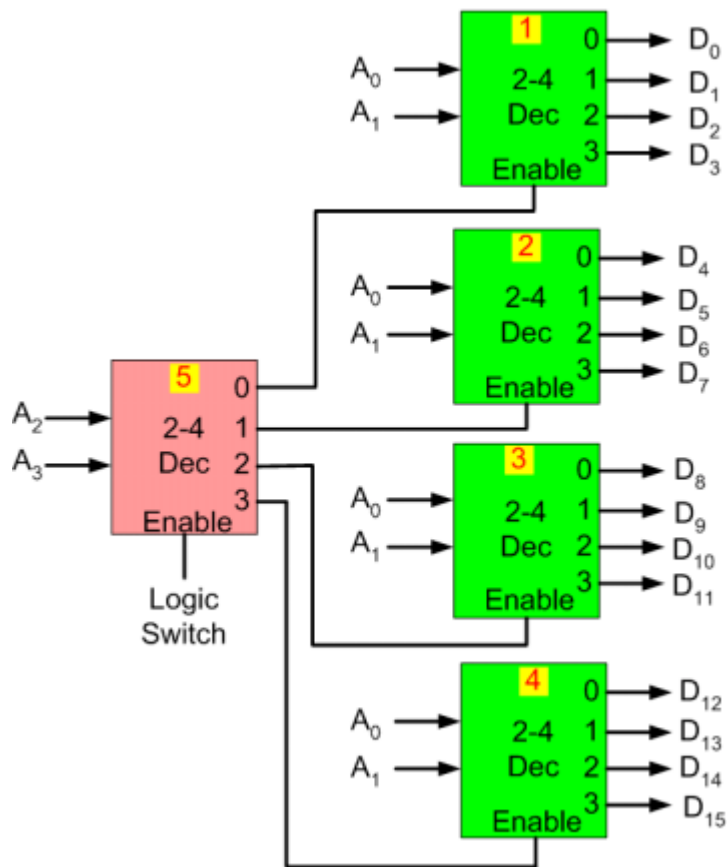


Figure: Constructing 4-to-16 decoder using 2-to-4 decoders

**Decoder example:** “Activate” line D2. The corresponding input combination that would activate this line is 0010. Now apply 00 at input of decoder ‘5’. This activates line ‘0’ connected to enable of decoder ‘1’. Once decoder ‘1’ is activated, inputs at  $A_1A_0 = 10$  activate line D2.

Thus we get the effect of a 4-16 decoder using this design, by applying input combinations in two steps.

As another example, to “activate” the line D10: The corresponding input combination is 1010. Apply 10 at the input of decoder ‘5’. This activates line ‘2’ connected to enable of decoder ‘3’. Once decoder ‘3’ is activated, the inputs at  $A_1A_0 = 10$  activate line D10.

Given two 4-bit unsigned numbers A and B, design a circuit which outputs the larger of the 2 numbers.

Here we will use Quad 2-1 Mux, and a 4-bit magnitude comparator. Both of these devices have been discussed earlier. The circuit is given in the figure. Since we are to select one of the two 4-bit numbers **A** ( $A_3A_2A_1A_0$ ) and **B** ( $B_3B_2B_1B_0$ ), it is obvious that we will need a quad 2-1 Mux. The inputs to this Mux are the two 4-bit numbers **A** and **B**. The select input of the Mux must be a signal which indicates the relative magnitude of the two numbers **A** and **B**. This signal may be True if  $A < B$  or if  $A > B$ . Such a signal is easily obtained from a 4-bit magnitude comparator.

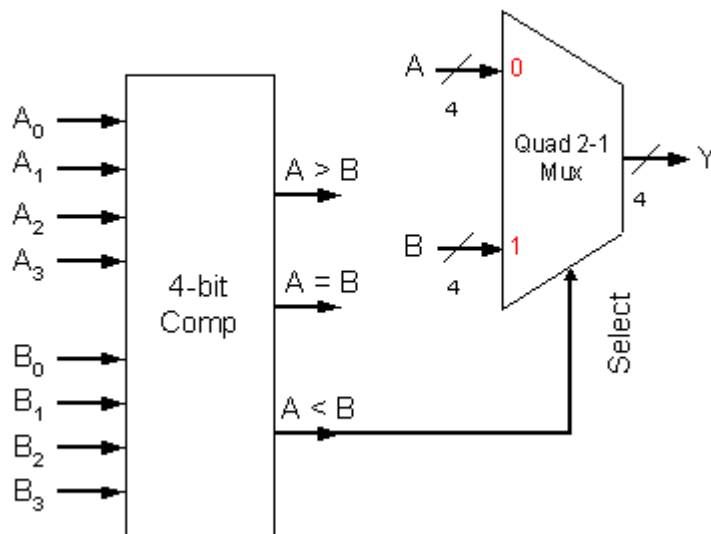


Figure : Circuit that outputs the larger of two numbers

By connecting the select input to the  $A < B$  output of the magnitude comparator, we must connect **A** to the **0** input of the Mux and **B** to the **1** input of the Mux. Alternatively, if we connect the select input to the  $A > B$  output of the magnitude comparator, we must connect **A** to the **1** input of the Mux and **B** to the **0** input of the Mux. In either case, the Mux output will be the larger of the two numbers.

### Designing a 16-bit adder using four 4-bit adders

Adds two 16-bit numbers **X** ( $X_0$  to  $X_{15}$ ), and **Y** ( $Y_0$  to  $Y_{15}$ ) producing a 16-bit Sum **S** ( $S_0$  to  $S_{15}$ ) and a carry out  $C_{16}$  as the most significant position. Thus, four 4-bit adders are connected in cascade. Each adder takes four bits of each input (**X** and **Y**) and generates a 4-bit sum and a carry that is fed into the next 4-bit adder as shown in Figure.

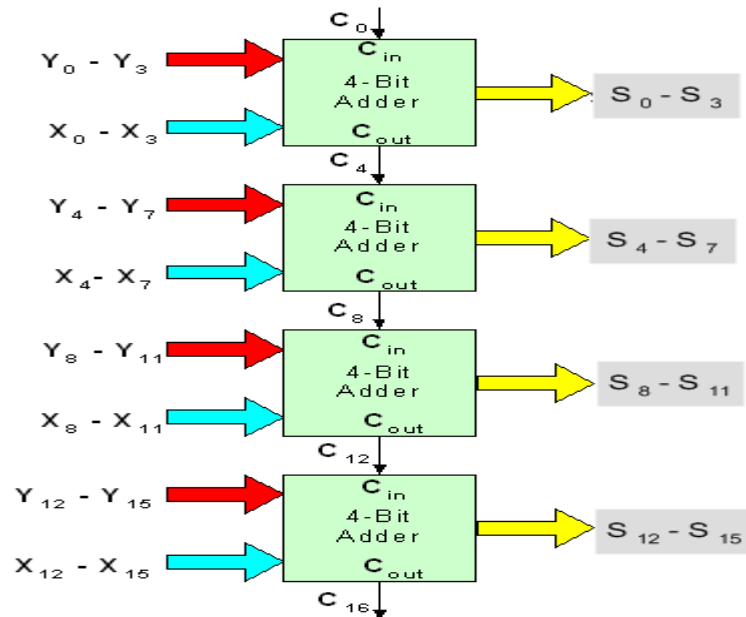


Figure : A 16-bit adder

### Designing an Excess-3 code converter using a Decoder and an Encoder

In this example, the circuit takes a BCD number as input and generates the corresponding

Ex-3 code. The truth table for this circuit is given in figure 6. The outputs 0000, 0001, 0010, 1101, 1110, and 1111 are never generated. To design this circuit, a 4-to-16 decoder and a 16-to-4 encoder are required. The design is given in figure 7. In this circuit, the decoder takes 4 bits as inputs, represented by variables  $w$ ,  $x$ ,  $y$ , and  $z$ . Based on these four bits, the corresponding min term output is activated. This decoder output then goes to the input of encoder which is three greater than the value generated by the decoder. The encoder then encodes the value and sends the output bits at  $A$ ,  $B$ ,  $C$ , and  $D$ . For example, suppose 0011 is sent as input. This will activate min term 3 of the decoder. This output is connected to input 6 of encoder. Thus the encoder will generate the corresponding bit combination, which is 0110.

W	X	Y	Z	A	B	C	D
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	0
1	0	0	1	1	1	0	0

Figure : table for BCD to Ex-3 conversion

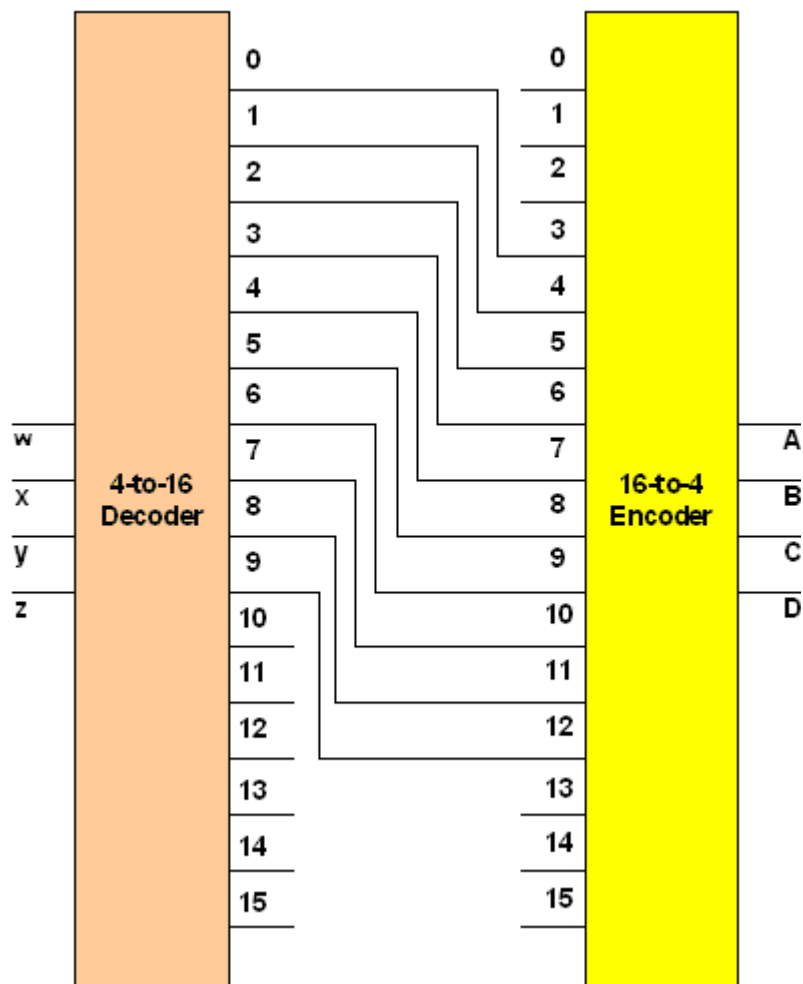


Figure : Circuit for BCD to Ex-3 conversion

**Recommended question and answer –unit-4**

**Jan 2009**

Q.4 b) The 1-bit comparator had 3 outputs corresponding ,to  $x > y$ ,  $x = y$  and  $x < y$ . It is possible to code these three outputs using two bits  $s_1 s_0$  such as  $s_1' s_0 = 00$ ,  $10$ ,  $01$  for  $x = y$ ,  $x > y$  and  $x < y$  respectively. This implies that only two-output lines occur from each 1-bit comparator. However at the output of the last 1-bit comparator, an additional network must be designed to convert the end results back to three outputs. Design such a 1-bit comparator as well as the output converter network.

Ans. :

x	y	$S_1$	$S_0$	G	E	L
0	0	0	0	0	1	0
0	1	0	1	0	0	1
1	0	1	0	1	0	0
1	1	0	0	0	1	0

K-map simplification :

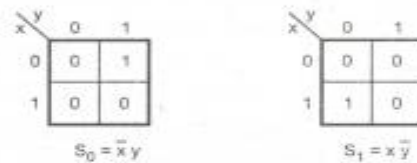


Fig. 4 (a)

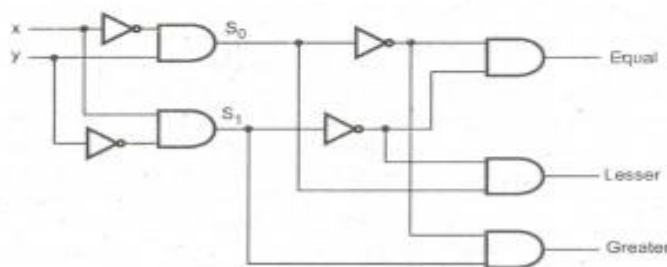


Fig. 4 (b)

**PART B**

**Jan-2008**

c) Realize  $F = \sum(x, y, z) = L(1, 2, 4, 5, 7)$  using 8 - to - 1 multiplexer (74L5151).  
(4)

Ans.: Equation =  $f(x, y, z) = L(1, 2, 4, 5, 7)$

Design table : We implement given equation SOP form using 8 : 1 MUX.

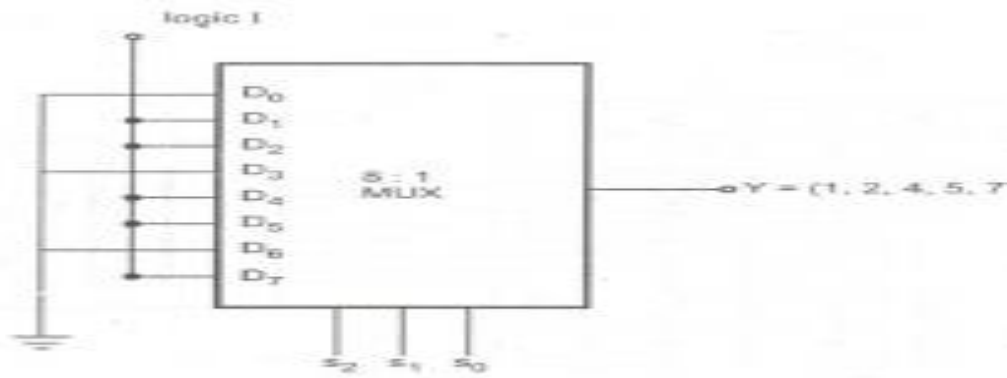


Fig. 13

**Aug 2009**

b) Implement the following Boolean function using 8 : 1 multiplexer.

$$f(A, B, C, D) = \bar{A} B \bar{D} + A C D + \bar{B} C D + \bar{A} \bar{C} D$$

Ans. : The given Boolean expression is not in standard SOP form. Let us first convert this in standard SOP form

$$\begin{aligned} F(A, B, C, D) &= \bar{A} B \bar{D} (C + \bar{C}) + A C D (B + \bar{B}) \\ &\quad + \bar{B} C D (A + \bar{A}) + \bar{A} \bar{C} D (B + \bar{B}) \\ &= \bar{A} B C \bar{D} + \bar{A} B \bar{C} \bar{D} + A B C D + A \bar{B} C D \\ &\quad + A \bar{B} C \bar{D} + \bar{A} \bar{B} C D + \bar{A} \bar{B} \bar{C} D + \bar{A} \bar{B} \bar{C} \bar{D} \\ &= \bar{A} B C \bar{D} + \bar{A} B \bar{C} \bar{D} + A B C D + A \bar{B} C D \\ &\quad + A \bar{B} C \bar{D} + \bar{A} \bar{B} C D + \bar{A} \bar{B} \bar{C} \bar{D} \end{aligned}$$

The truth table for this standard SOP form can be given as

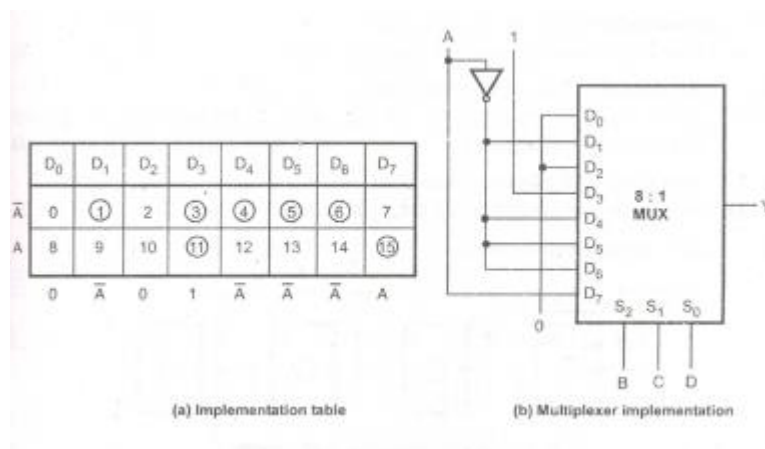
Sr. No.	Minterms	A	B	C	D	Y
0.		0	0	0	0	0
1.	$\bar{A} \bar{B} \bar{C} D$	0	0	0	1	1
2.		0	0	1	0	0
3.	$\bar{A} \bar{B} C D$	0	0	1	1	1
4.	$\bar{A} \bar{B} C \bar{D}$	0	1	0	0	1
5.	$\bar{A} \bar{B} \bar{C} D$	0	1	0	1	1
6.	$\bar{A} \bar{B} C \bar{D}$	0	1	1	0	1
7.		0	1	1	1	0

8.		1	0	0	0	0
9.		1	0	0	1	0
10.		1	0	1	0	0
11.	A B C D	1	0	1	1	1
12.		1	1	0	0	0
13.		1	1	0	1	0
14.		1	1	1	0	0
15.	A B C D	1	1	1	1	1

Table 2 Truth table

From the truth table Boolean function can be implemented using 8: 1 multiplexer

as follows:



**Aug-2008**

Q.3 a) Realize the following Boolean function  $f(A, B, C, D) = \sum(0, 1, 3, 5, 7)$

Using - i) 8 : 1 MUX (74151) ii) 4 : 1 MUX (74153).

Ans. : Design using 8 : 1 MUX :

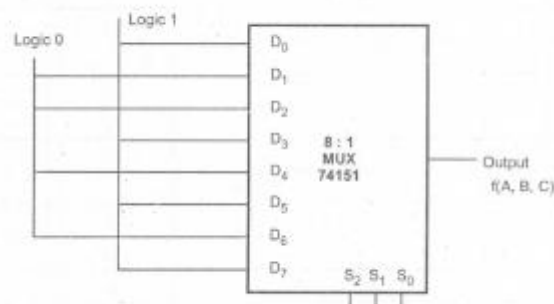


Fig. 5 (a)

b) Design a combinational logic circuit that will convert a straight BCD digit to an Excess - 3 BCD digits.

i) Construct the truth table. ii) Simplify each output function using Karnaugh map.

map. and write the reduced equations. .iii) Draw the resulting logic diagram. (12)

Ans. : i) Truth table

Decimal	BCD inputs				Excess-3 outputs			
	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>	E <sub>3</sub>	E <sub>2</sub>	E <sub>1</sub>	E <sub>0</sub>
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

ii) Simplified equations using K-map

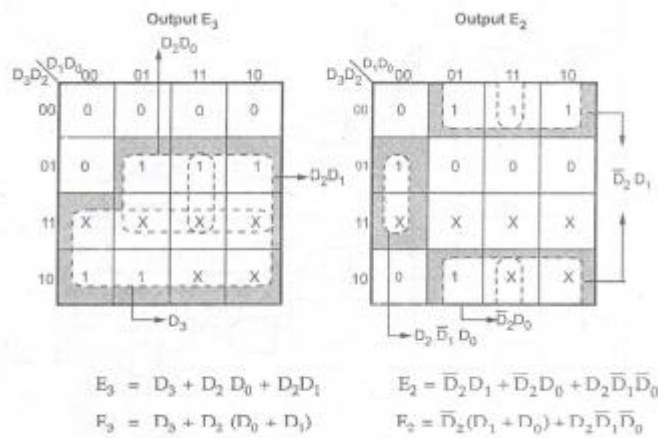


Fig. 6 (a)

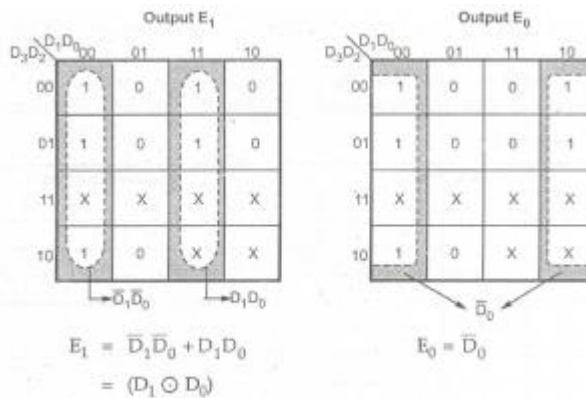


Fig. 6 (b)



iii) Resulting logic diagram

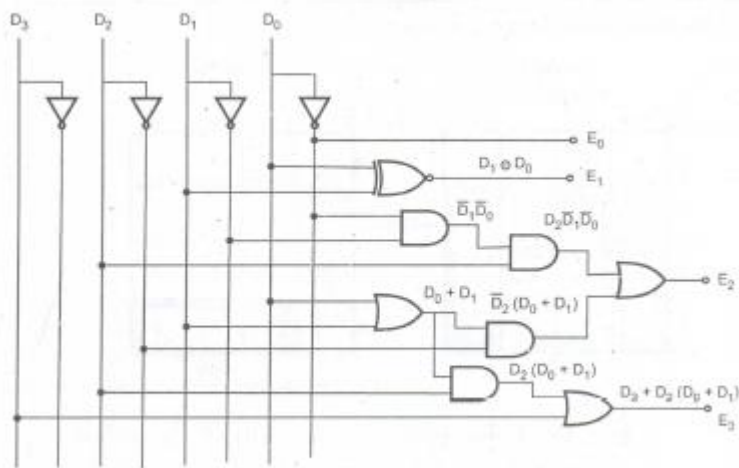


Fig. 6 (c) BCD to Excess-3 code converter

notes4free.in

**MODULE-3****Sequential Circuits – 1:****7 Hours**

Basic Bistable Element, Latches, SR Latch, Application of SR Latch, A Switch Debouncer, The S R Latch, The gated SR Latch, The gated D Latch, The Master-Slave Flip-Flops (Pulse-Triggered Flip-Flops): The Master-Slave SR Flip-Flops, The Master-Slave JK Flip-Flop, Edge Triggered Flip-Flop: The Positive Edge-Triggered D Flip-Flop, Negative-Edge Triggered D Flip-Flop

**Recommended readings:**

1. Donald D Givone, “Digital Principles and Design “, Tata McGraw  
Hill Edition, 2002.  
Unit - 6.1, 6.2, 6.4, 6.5

**Introduction :**

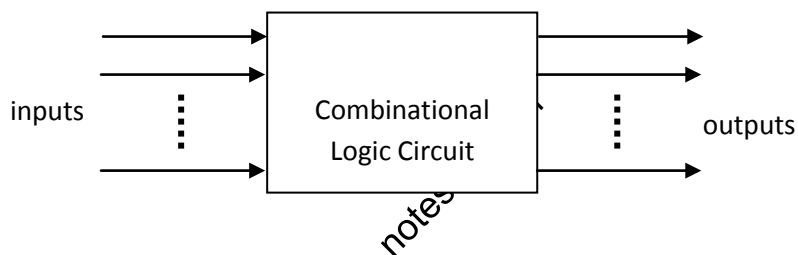
Logic circuit is divided into two types.

1. Combinational Logic Circuit
2. Sequential Logic Circuit

**Definition :**

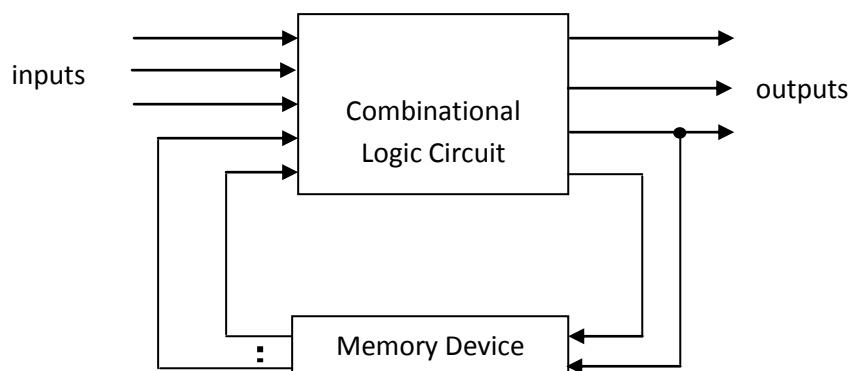
1. Combinational Logic Circuit :

The circuit in which outputs depends on only present value of inputs. So it is possible to describe each output as function of inputs by using Boolean expression. No memory element involved. No clock input. Circuit is implemented by using logic gates. The propagation delay depends on, delay of logic gates. Examples of combinational logic circuits are : full adder, subtractor, decoder, codeconverter, multiplexers etc.



2. Sequential Circuits :

Sequential Circuit is the logic circuit in which output depends on present value of inputs at that instant and past history of circuit i.e. previous output. The past output is stored by using memory device. The internal data stored in circuit is called as state. The clock is required for synchronization. The delay depends on propagation delay of circuit and clock frequency. The examples are flip-flops, registers, counters etc.



- Basic Bistable element.
  - Flip-Flop is Bistable element.

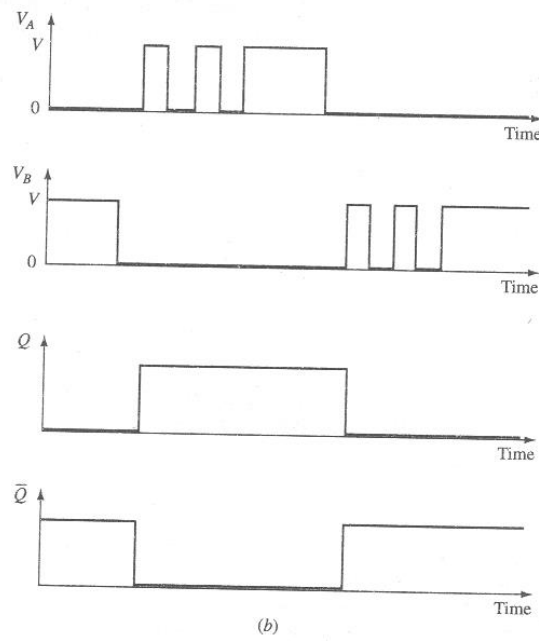
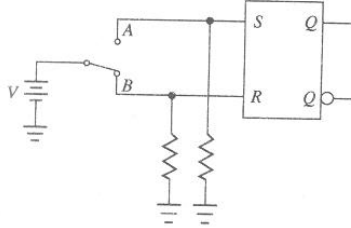
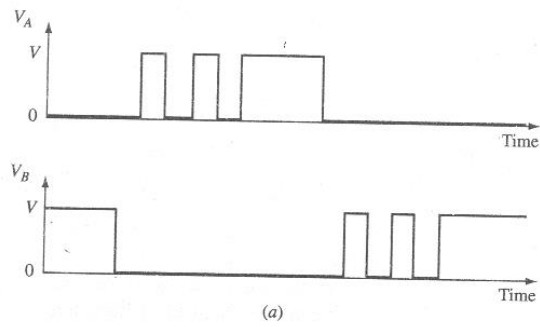
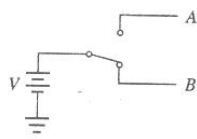
- It consist of two cross coupled NOT Gates.
- It has two stable states.
- Q and  $\bar{Q}$  are two outputs complement of each other.
- The data stored 1 or 0 in basic bistable element is state of flip-flop.
- 1 – State is set condition for flip-flop.
- 0 – State is reset / clear for flip-flop.
- It stores 1 or 0 state as long power is ON.

**Latches :****S-R Latch : Set-reset Flip-Flop**

- Latch is a storage device by using Flip-Flop.
- Latch can be controlled by direct inputs.
- Latch outputs can be controlled by clock or enable input.
- Q and  $\bar{Q}$  are present state for output.
- $Q^+$  and  $\bar{Q}^+$  are next states for output.
- The function table / Truth table gives relation between inputs and outputs.
- The S=R=1 condition is not allowed in SR FF as output is unpredictable.

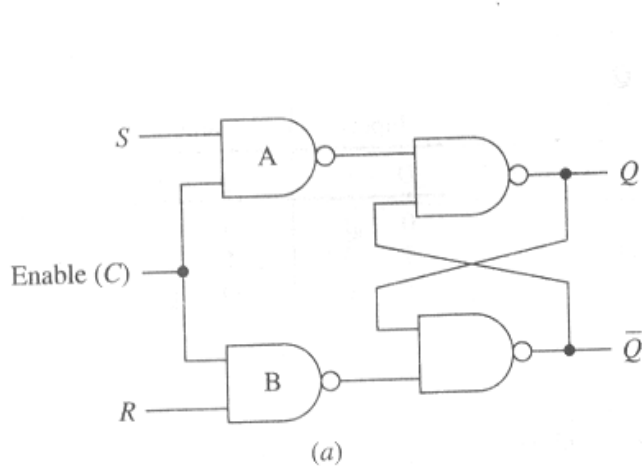
Application of SR Latch :

- A switch debouncer



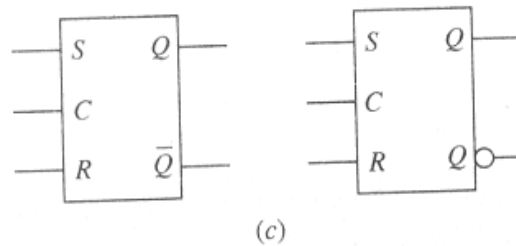
- Bouncing problem with Push button switch.
- Debouncing action.
- SR Flip-Flop as switch debouncer.

**Gated SR Latch :**

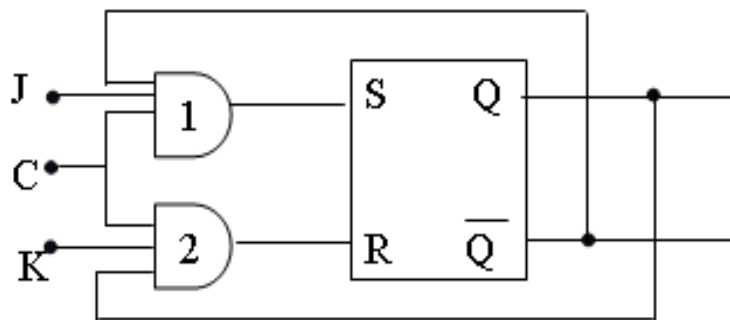






Inputs			Outputs	
S	R	C	Q <sup>+</sup>	Q̄ <sup>+</sup>
0	0	1	Q	Q̄
0	1	1	0	1
1	0	1	1	0
1	1	1	1*	1*
X	X	0	Q	Q̄

\*Unpredictable behavior will result if S and R return to 0 simultaneously or C returns to 0 while S and R are 1



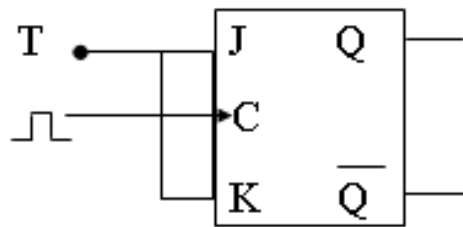
- Enable input C is clock input.
- C=1, Output changes as per input condition.
- C=0, No change of state.
- S=1, R=0 is set condition for Flip-flop.
- S=0, R=1 is reset condition for Flip-flop.
- S=R=1 is ambiguous state, not allowed.

**JK Flip-Flop by using SR Flip-Flop**

<b>Function Table</b>					
Input			Output		
C	J	K	$Q^+$	$\bar{Q}^+$	Remark
	0	0	Q	$\bar{Q}$	NC
	0	1	0	1	Reset
	1	0	1	0	Set
	1	1	$\bar{Q}$	Q	Toggle
0	x	x	Q	$\bar{Q}$	NC

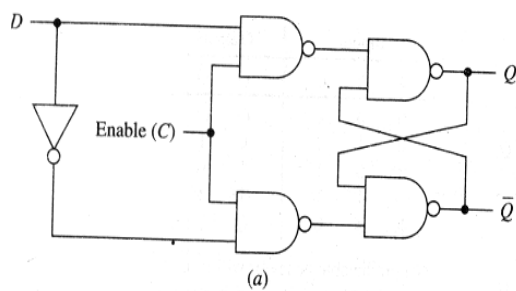
In SR FF,  $S=R=1$  condition is not allowed.

- JK FF is modified version of SR FF.
- Due to feedback from output to input AND Gate  $J=K=1$  is toggle condition for JK FF.
- The output is complement of the previous output.
- This condition is used in counters.
- T-FF is modified version of JK FF in which  $T=J=K=1$ .

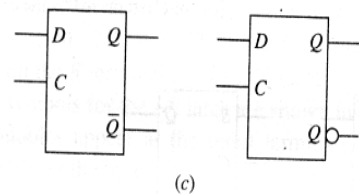


Function Table				
Input		Output		
C	T	$Q^+$	$\bar{Q}^+$	Remark
0	0	Q	$\bar{Q}$	NC
0	1	$\bar{Q}$	Q	Toggle
1	x	Q	$\bar{Q}$	NC

**Gated D Latch :**



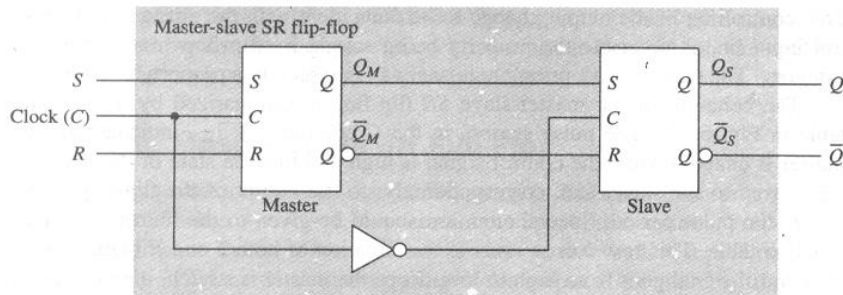
Inputs		Outputs	
D	C	$Q^+$	$\bar{Q}^+$
0	1	0	1
1	1	1	0
X	0	Q	$\bar{Q}$



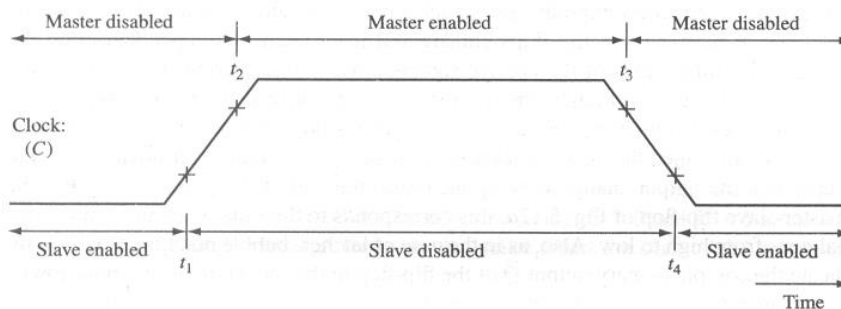
- D Flip-Flop is Data Flip-Flop.
- D Flip-Flop stores 1 or 0.
- R input is complement of S.
- Only one D input is present.
- D Flip-Flop is a storage device used in register.



Master slave SR Flip-Flop



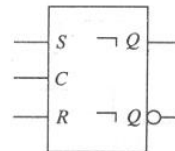
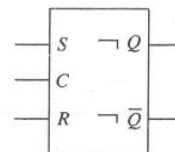
(a)



(b)

Inputs			Outputs	
S	R	C	Q <sup>+</sup>	Q̄ <sup>+</sup>
0	0		Q	Q̄
0	1		0	1
1	0		1	0
1	1		Undefined	Undefined
X	X	0	Q	Q̄

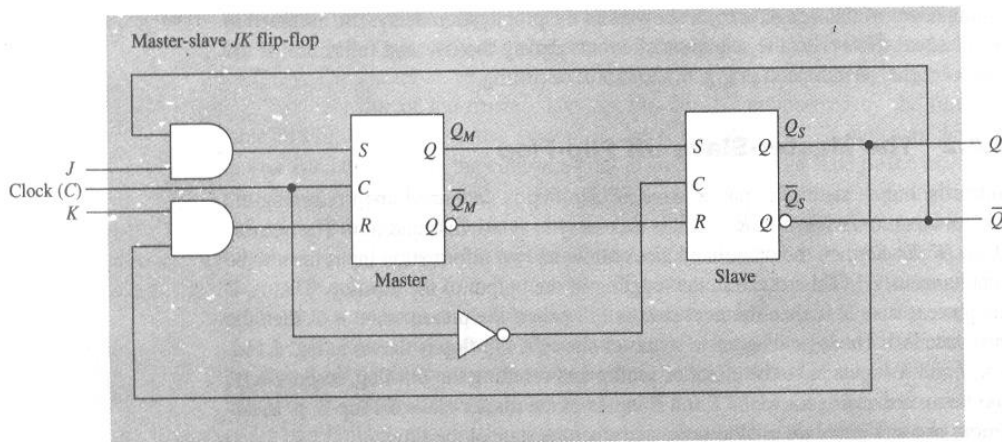
(c)



(d)

- Two SR Flip-Flop, 1<sup>st</sup> is Master and 2<sup>nd</sup> is slave.
- Master Flip-Flop is positive edge triggered.
- Slave Flip-Flop is negative edge triggered.
- Slave follows master output.
- The output is delayed.

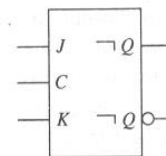
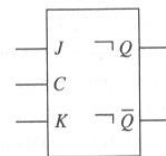
Master slave JK Flip-Flop



(a)

Inputs			Outputs	
J	K	C	Q <sup>+</sup>	Q <sup>-</sup>
0	0		Q	Q <sup>-</sup>
0	1		0	1
1	0		1	0
1	1		Q <sup>-</sup>	Q
X	X	0	Q	Q <sup>-</sup>

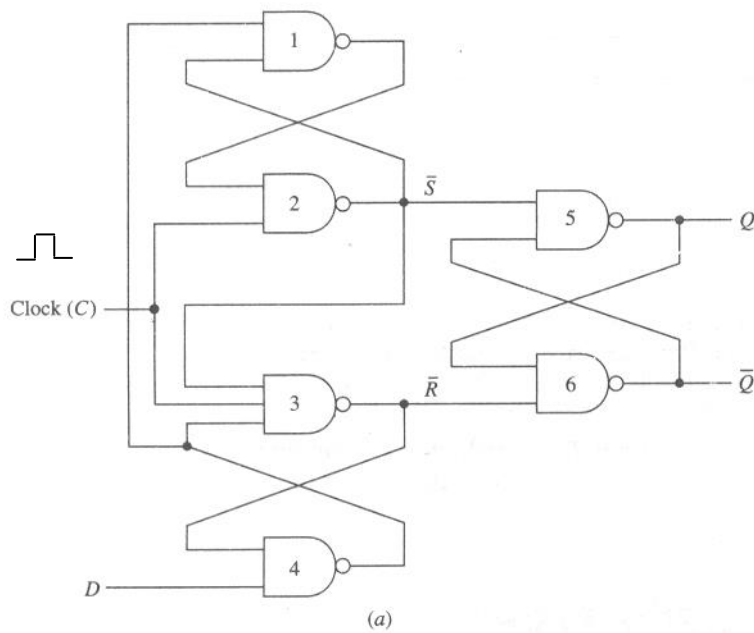
(b)



(c)

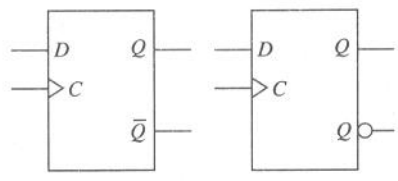
- In SR Flip-Flop the input combination S=R=1 is not allowed.
- JK FF is modified version of SR FF.
- Due to feedback from slave FF output to master, J=K=1 is allowed.
- J=K=1, toggle, action in FF.
- This finds application in counter.

Positive Edge Triggered D Flip-Flop



Inputs		Outputs	
D	C	Q <sup>+</sup>	Q̄ <sup>+</sup>
0	↑	0	1
1	↑	1	0
X	0	Q	Q̄
X	1	Q	Q̄

(b)



(c)

notes4free.

- When C=0, the output of AND Gate 2 & 3 is equal to 1.  
 $\bar{S} = \bar{R} = 1$ , No Change of State
- If C=1, D=1, the output of AND Gate 2 is 0 and 3 is 1.  
 $\bar{S} = 0, \bar{R} = 1$ , Q = 1 and  $\bar{Q} = 0$

**Recommended question and answer –unit-5****Jan 2009**

Q.6 a) Design a 4-bit universal shift register using positive edge triggered D flip-flops to operate as shown in the table .

Select line		Data line selected	Register operation
$S_0$	$S_1$		
0	0	$i_0$	HOLD
0	1	$i_1$	Shift RIGHT
1	0	$i_2$	Shift LEFT
1	1	$i_3$	Parallel load

(12)

Ans. : Universal shift register: A register capable of shifting in one direction only is a unidirectional shift register. A register capable of shifting in both directions is a bidirectional shift register. If the register has both shifts (right shift and left shift) and parallel load capabilities, it is referred to as Universal shift register. The Fig. 5 (See next page) shows the 4-bit universal shift register. It has all the capabilities listed above. It consists of four flip-flops and four multiplexers. The four multiplexers have two common selection inputs  $S_1$  and  $S_0$  and they select appropriate input for each flip-flop. The Table 1 shows the register operation depending on the selection inputs of multiplexers. When  $S_1 S_0 = 00$ , input 0 is selected and the present value of the register is applied to the D inputs of the flip-flops. This results no change in the register value. When  $S_1 S_0 = 01$ , input 1 is selected and circuit connections are such that it operates as a right shift register. When  $S_1 S_0 = 10$ , input 2 is selected and circuit connections are such that it operates as a left shift register. Finally, when  $S_1 S_0 = 11$ , the binary information on the parallel input lines is transferred into the register simultaneously and it is a parallel load operation.

Mode control		Register operation
$S_1$	$S_0$	
0	0	No change
0	1	Shift right
1	0	Shift left
1	1	Parallel load

Table 1 Mode control and register operation

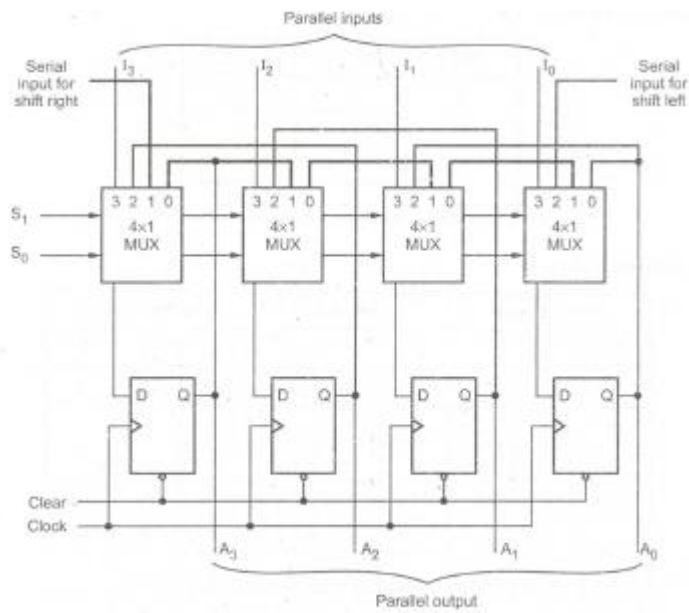


Fig. 5 4-Bit universal shift register

**Jan -2008**

5. What is the significance of edge triggering ? Explain the working of edge triggered D-flip-flop and T-flip-flop with their functional table. (6)

Ans. : For the edge triggered FF, it is necessary to apply the clock signal in the form of sharp positive and negative spikes instead of in the form of pulse train. These spikes can be derived from the rectangular clock pulses with the help of a passive differentiator as shown in Fig. 14. Edge triggered D Flip-Flop Fig. 15 shows the edge triggered DFF. It consists of gated D latch and a differentiator circuit. The clock pulses are applied to the circuit through a differentiator formed by RC and a rectifier circuit consisting of diode D and R2. The NAND gates 1 through 5 form a D latch. The differentiator converts the clock pulse into positive and negative spikes as shown in the Fig. 16 and the combination of D and R2 will allow only the positive spikes to pass through blocking the negative spikes.

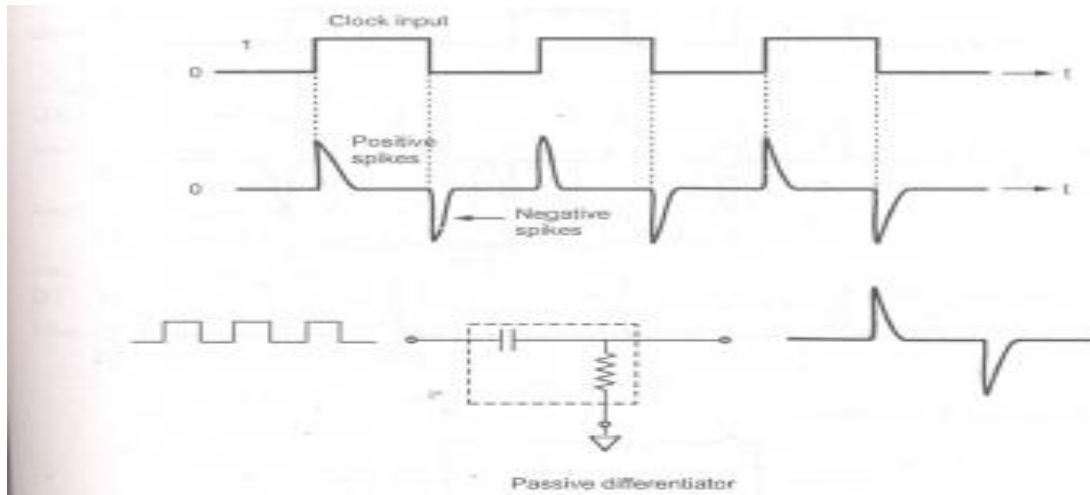


Fig. 14 Use of differentiator to obtain sharp edges

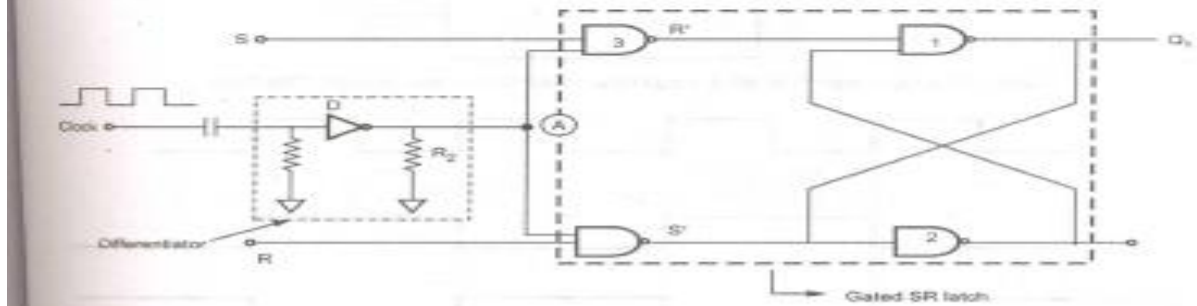


Fig. 15

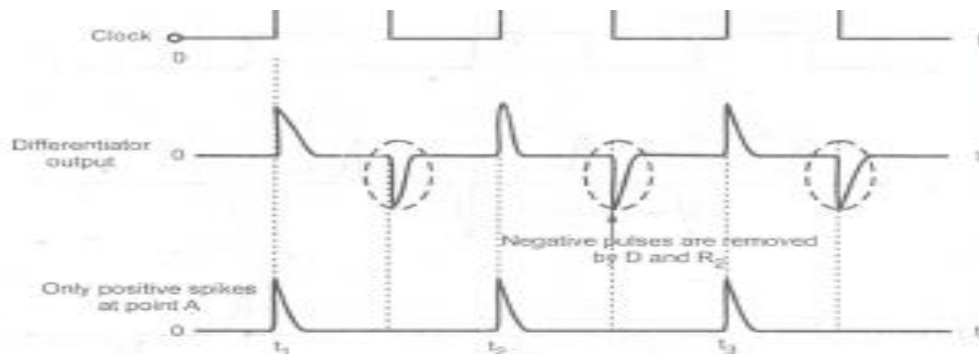
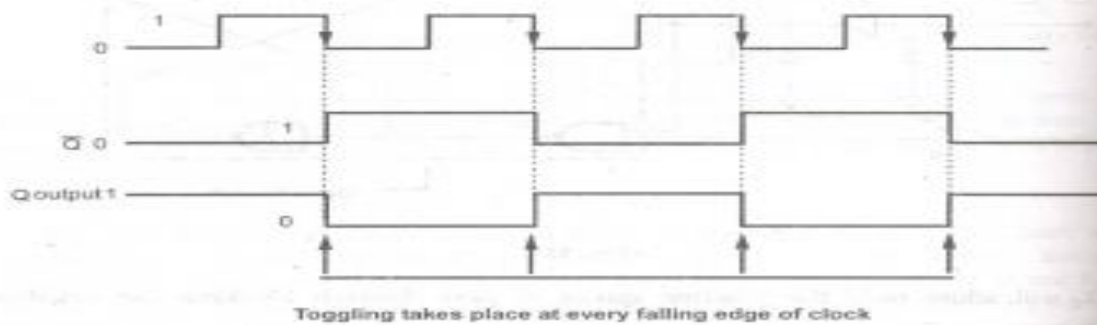


Fig. 16

Edge triggered T flip-flop



Fig. 17 Logic symbol of a negative edge triggered toggle flip-flop



Aug 2009

Q.5 a) Clearly distinguish between

- i) Synchronous and asynchronous circuits.
- ii) Combinational and sequential circuits.

Ans. : i) Synchronous and asynchronous circuits :

Sr. No.	Synchronous sequential circuits	Asynchronous sequential circuits
1.	In synchronous circuits, memory elements are clocked flip-flops.	In asynchronous circuits, memory elements are either unlocked flip-flops or time delay elements.
2.	In synchronous circuits, the change in input signals can affect memory element upon activation of clock signal.	In asynchronous circuits change in input signals can affect memory element at any instant of time.
3.	The maximum operating speed of clock depends on time delays involved.	Because of absence of clock, asynchronous circuits can operate faster than synchronous circuits.
4.	Easier to design.	More difficult to design.

b) Explain the working of 4-bit asynchronous counter.

Ans. : 4-bit asynchronous counter :

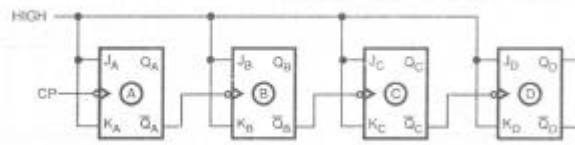


Fig. 10 4-bit asynchronous down counter

- 1) 4 flip-flops are employed to create a 4-bit asynchronous counter as shown.
  - 2) The clock signal is connected to the clock input of only first stage flip-flop.
  - 3) Because of the inherent propagation delay time through a flip-flop, two flip-flops never trigger simultaneously. Thus, it works in an asynchronous operation.
  - 4) Output of the first flip-flop triggers the second flip-flop and so on.
- S) At the output of flip-flops, we get the counted value of the counter.

c) Explain Johnson counter with its circuit diagram and timing diagram. (8)

Ans. : 4-bit Johnson counter :

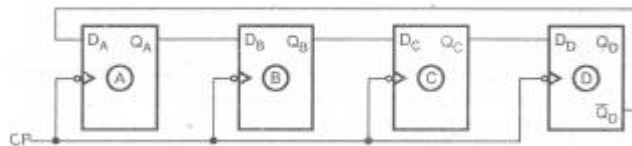


Fig. 11 Four-bit Johnson counter

- 1) Initially, the register is cleared.  
∴ all the outputs  $Q_A, Q_B, Q_C, Q_D$  are zero.
- 2) The complement of  $Q_0$  is 1 which is connected back to the D input of first stage.  
∴  $D_A$  is, 1.  
∴ The output becomes  $Q_A = 1, Q_B = 0, Q_C = 0$  and  $Q_D = 0$ .
- 3) The next clock pulse produces  $Q_A = 1, Q_B = 1, Q_C = 0$  and  $Q_D = 0$ .

The sequence is' given as :



Clock Pulse	Q <sub>A</sub>	Q <sub>B</sub>	Q <sub>C</sub>	Q <sub>D</sub>
0	0	0	0	0
1	1	0	0	0
2	1	1	0	0
3	1	1	1	0
4	1	1	1	1
5	0	1	1	1
6	0	0	1	1
7	0	0	0	1

Table 3 Four-bit Johnson sequence

**Aug-2008**

b) Fig. 2 shows a BCD counter that produces a 4-bit cutput representing code for the number of pulses that have been applied to the counter  $i$ ; example, after four pulses have occurred, the counter ouq (ABeD) = (0100h = (04)10'. The counter resets to 0000 on the tenth] starts counting over again. Design the logic circuit that produces a HIGH Whenever the count is 2, 3 or 9. Use K - mapping and take advai

"don't care" conditions. Implement the logic circuit using NAND gates.

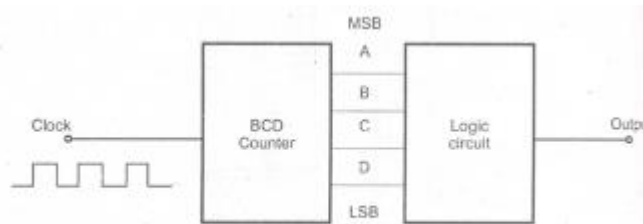


Fig. 2

Ans. :

Input clock	BCD counter output				Output
	A	B	C	D	
1	0	0	0	0	0
2	0	0	0	1	0

3	0	0	1	0	1
4	0	0	1	1	1
5	0	1	0	0	0
6	0	1	0	1	0
7	0	1	1	0	0
8	0	1	1	1	0
9	1	0	0	0	0
10	1	0	0	1	1
11	1	0	1	0	X
12	1	0	1	1	X
13	1	1	0	0	X
14	1	1	0	1	X
15	1	1	1	0	X
16	1	1	1	1	X

BCD numbers are present in between 0 to 9 so from 9 to 15 we take don't care condition and when 2, 3, or 9 detected we take output as 1.

Output =  $AD + BC$

K-map simplification

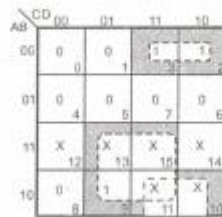


Fig. 2 (a)

Implementation using NAND gate :

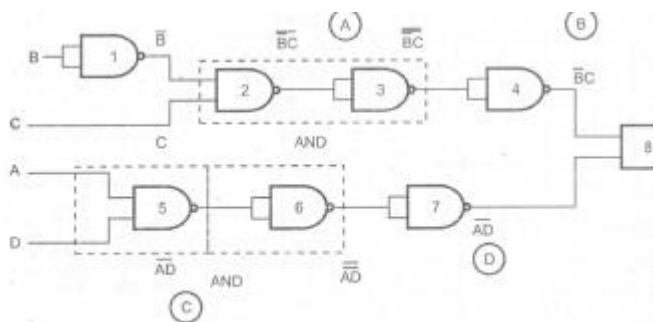
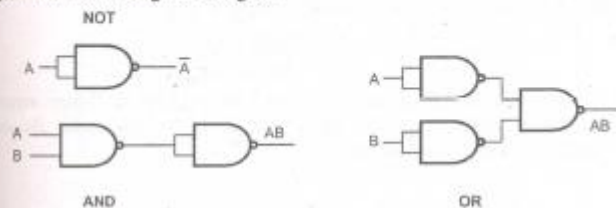


Fig. 2 (c)

We get same data or signal at point(A) and (B) as well as point(C) and (D) remove gate 2, 3 for at point (A), (B) and 6, 7, for point (C), (D).

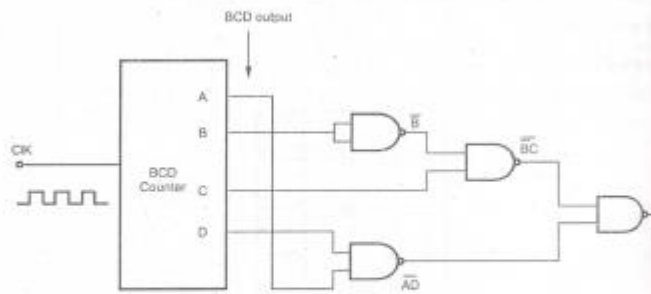


Fig. 2 (d)

Output will be the 1 when output of BCD  $(0010)_2 = (2)_{10}$   
 $(0011)_2 = (3)_{10}$   
 $(1001)_2 = (9)_{10}$

**Aug -2008**

**Q.4 a)** Design a 4 - bit BCD adder circuit using 7483 IC chip, with self correcting circuit i.e., a provision has. to be made in the circuit, in case if the sum of the BCD number exceeds 9. (12)

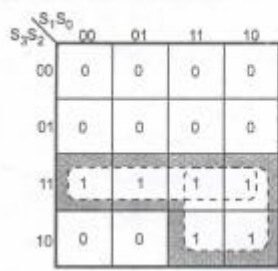
Ans. : Truth table is given as ,

Inputs				Output
S <sub>3</sub>	S <sub>2</sub>	S <sub>1</sub>	S <sub>0</sub>	Y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Sum is valid BCD number  
 $\therefore Y = 0$

invalid BCD number  
 $\therefore Y = 1$

K-map simplification



$\therefore Y = S_3S_2 + S_3S_1$

Circuit diagram for 4-bit BCD adder :

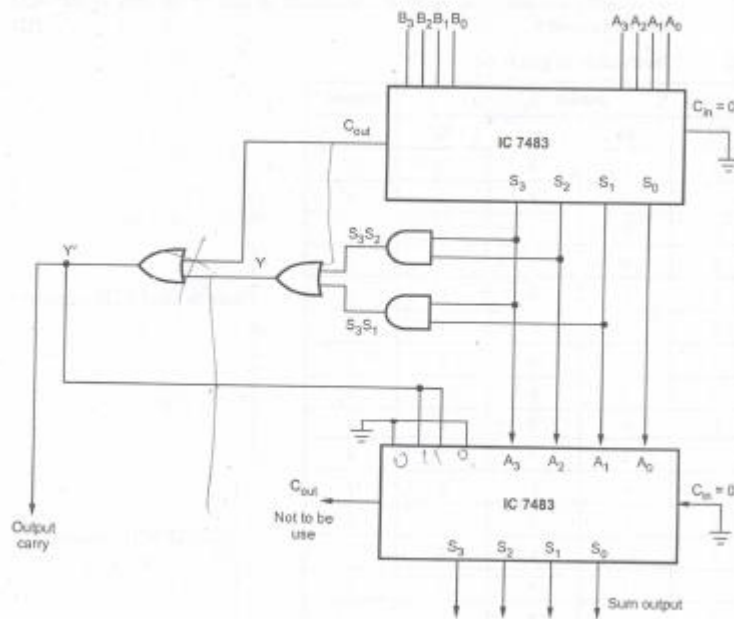


Fig. 7 (b)

Aug-2007

notes4free.in

b) Implement the following Boolean function using 8:1 MUX :  
 $F(A, B, C, D) = \Sigma m(1, 2, 5, 9, 10, 14)$

Sol. : Implementation Table :

	D <sub>0</sub>	D <sub>1</sub>	D <sub>2</sub>	D <sub>3</sub>	D <sub>4</sub>	D <sub>5</sub>	D <sub>6</sub>	D <sub>7</sub>	
$\bar{C}$	0	①	②	3	4	⑤	6	7	← Minterms for C = 0
C	8	⑨	⑩	11	12	13	⑭	15	← Corresponding minterms for C = 1
	0	1	1	0	0	$\bar{C}$	C	0	

Fig. 10

Multiplexer Implementation :

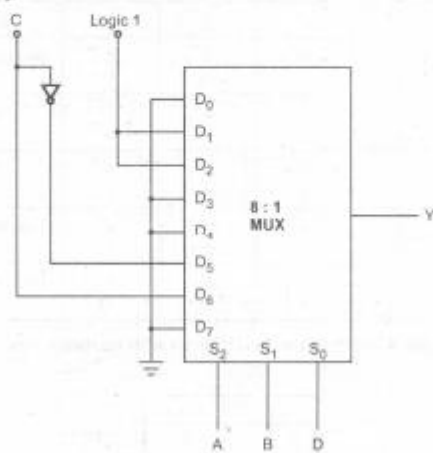


Fig. 11

**MODULE 4:****Hours-10****Sequential Circuits – 2:**

Characteristic Equations, Registers, Counters -Binary Ripple Counters, Synchronous Binary counters, Counters based on Shift Registers, Design of a Synchronous counters, Design of a Synchronous Mod-6 Counter using clocked JK Flip-Flops Design of a Synchronous Mod-6

Counter using clocked D, T, or SR Flip-Flops

notes4free.in

**Recommended readings:**

1. Donald D Givone, “Digital Principles and Design “, Tata McGraw

Hill Edition, 2002.

Unit - 6.6, 6.7, 6.8,6.9 – 6.9.1 and 6.9.2

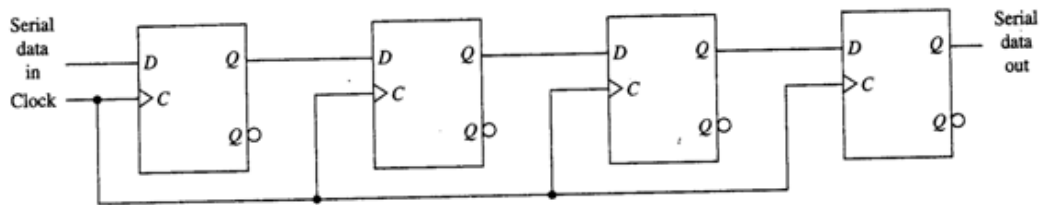
REGISTERS

Fig. : Serial-In, Serial-Out Unidirectional Shift Register

- Register is a group of Flip-Flops.
- It stores binary information 0 or 1.
- It is capable of moving data left or right with clock pulse.
- Registers are classified as
  - Serial-in Serial-Out
  - Serial-in parallel Out
  - Parallel-in Serial-Out
  - Parallel-in parallel Out

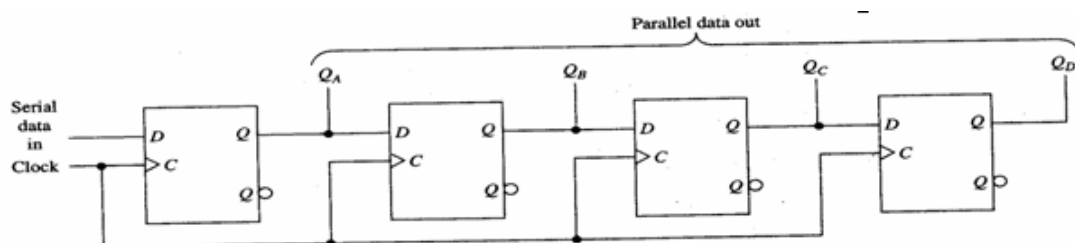


Fig. : Serial-In, Parallel-Out Unidirectional Shift Register

## Parallel-in Unidirectional Shift Register

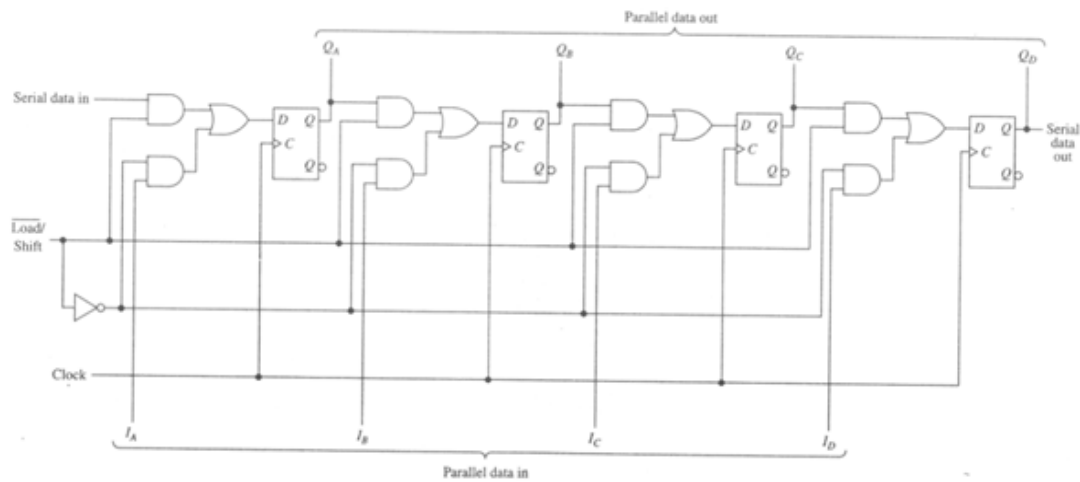
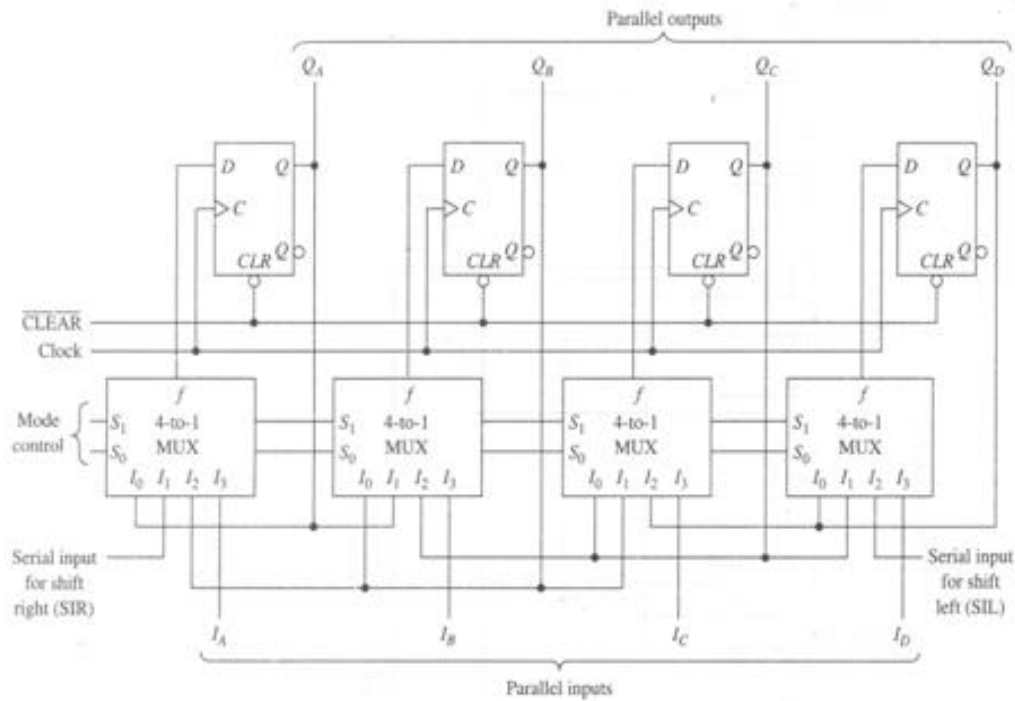


Fig. : Parallel-in Unidirectional Shift Register

notes4free.in

- Parallel input data is applied at  $I_A I_B I_C I_D$ .
- Parallel output  $Q_A Q_B Q_C Q_D$ .
- Serial input data is applied to A FF.
- Serial output data is at output of D FF.
- $\bar{L}/\text{Shift}$  is common control input.
- $\bar{L}/S = 0$ , Loads parallel data into register.
- $\bar{L}/S = 1$ , shifts the data in one direction.

Universal Shift Register



Select lines		Register operation
$S_1$	$S_0$	
0	0	Hold
0	1	Shift right
1	0	Shift left
1	1	Parallel load

- Bidirectional Shifting.
- Parallel Input Loading.
- Serial-Input and Serial-Output.
- Parallel-Input and Serial-Output.
- Common Reset Input.
- 4:1 Multiplexer is used to select register operation.



### COUNTERS

- Counter is a register which counts the sequence in binary form.
- The state of counter changes with application of clock pulse.
- The counter is binary or non-binary.
- The total no. of states in counter is called as modulus.
- If counter is modulus-n, then it has n different states.
- State diagram of counter is a pictorial representation of counter states directed by arrows in graph.

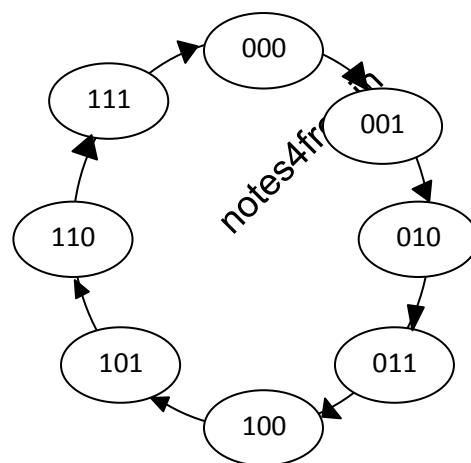


Fig. State diagram of mod-8 counter

### Ripple and Arbitrary Counters

In this lesson, you will learn about:

- \_ Ripple Counters
- \_ Counters with arbitrary count sequence

Design of ripple Counters

Two types of counters are identifiable:

- \_ *Synchronous* counters, which have been discussed earlier, and
- \_ *Ripple* counters.

In ripple counters, flip-flop output transitions serve as a source for triggering other flipflops.

In other words, clock inputs of the flip-flops are triggered by output transitions of other flip-flops, rather than a common clock signal.

Typically, T flip-flops are used to build ripple counters since they are capable of complementing their content (See Figure 1).

The signal with the pulses to be counted, i.e. "*Pulse*", is connected to the clock input of the flip-flop that holds the **LSB** (FF # 1).

The output of each FF is connected to the clock input of the next flip-flop in sequence.

The flip-flops are negative edge triggered (bubbled clock inputs).

$T=1$  for all FFs ( $J = K = 1$ ). This means that each flip-flop complements its value if C input goes through a negative transition (1  $\rightarrow$  0).

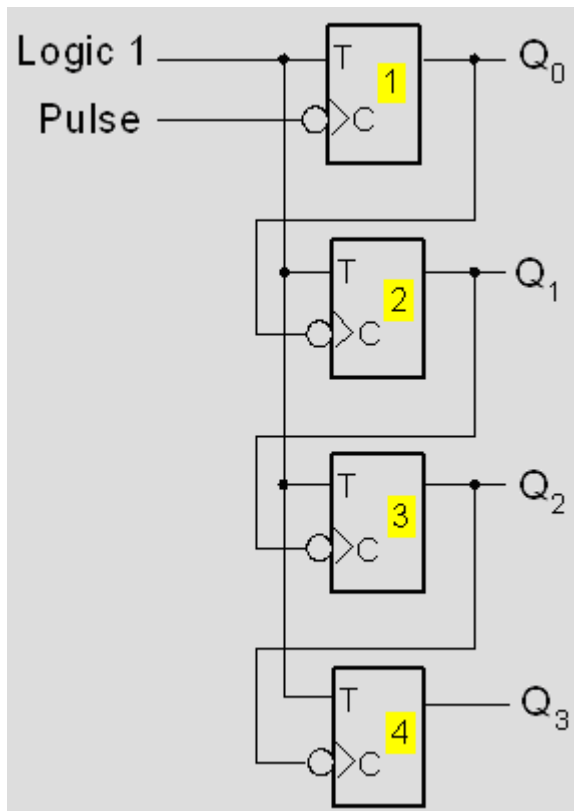


Figure 1: A ripple counter

The previous ripple up-counter can be converted into a down-counter in one of two ways:

- \_ Replace the negative-edge triggered FFs by positive-edge triggered FFs, or
- \_ Instead of connecting C input of FF  $Q_i$  to the output of the preceding FF ( $Q_{i-1}$ ) connect it to the complement output of that FF ( $Q_{i-1}$ ).

#### Advantages of Ripple Counters:

- \_ simple hardware and design.

#### Disadvantages of Ripple Counters:

- \_ They are asynchronous circuits, and can be unreliable and delay dependent, if more logic is added.
- \_ Large ripple counters are slow circuits due to the length of time required for the ripple to occur.

### Counters with Arbitrary Count Sequence:

Design a counter that follows the count sequence: 0, 1, 2, 4, 5, 6. This counter can be designed with any flip-flop, but let's use the JK flip-flop.

Notice that we have two “*unused*” states (3 and 7), which have to be dealt with (see Figure 2). These will be marked by don't cares in the state table (Refer to the design of sequential circuits with unused states discussed earlier). The state diagram of this counter is shown in Figure 2.

In this figure, the unused states can go to any of the valid states, and the circuit can continue to count correctly. One possibility is to take state 7 (111) to 0 (000) and state 3 (011) to 4 (100).

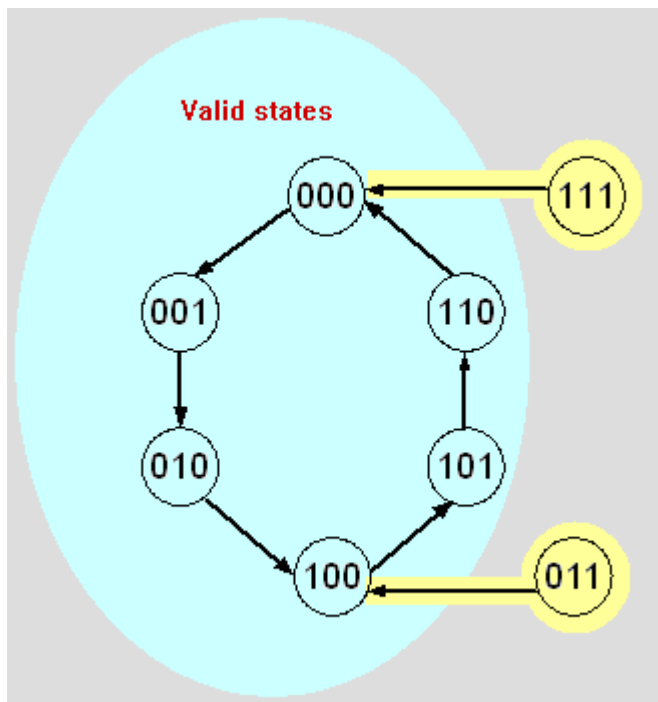


Figure 2: State diagram for arbitrary counting sequence

The design approach is similar to that of synchronous circuits. The state transition table is built as shown in Figure 3 and the equations for all J and K inputs are derived. Notice that we have used don't care for the unused state (although we could have used 100 as the next state for 011, and 000 as the next state of 111).

Unused states

Present State			Next State			Flip-Flop Inputs					
A	B	C	A	B	C	J <sub>A</sub>	K <sub>A</sub>	J <sub>B</sub>	K <sub>B</sub>	J <sub>C</sub>	K <sub>C</sub>
0	0	0	0	0	1	0	X	0	X	1	X
0	0	1	0	1	0	0	X	1	X	X	1
0	1	0	1	0	0	1	X	X	1	0	X
0	1	1	X	X	X	X	X	X	X	X	X
1	0	0	1	0	1	X	0	0	X	1	X
1	0	1	1	1	0	X	0	1	X	X	1
1	1	0	1	1	1	X	0	X	0	1	X
1	1	1	X	X	X	X	X	X	X	X	X

Figure 3: State table for arbitrary counting sequence

The computed J and K input equations are as follows:

$$J_A = B \quad K_A = \bar{B}$$

$$J_B = C \quad K_B = 1$$

$$J_C = \bar{B} \quad K_C = 1$$

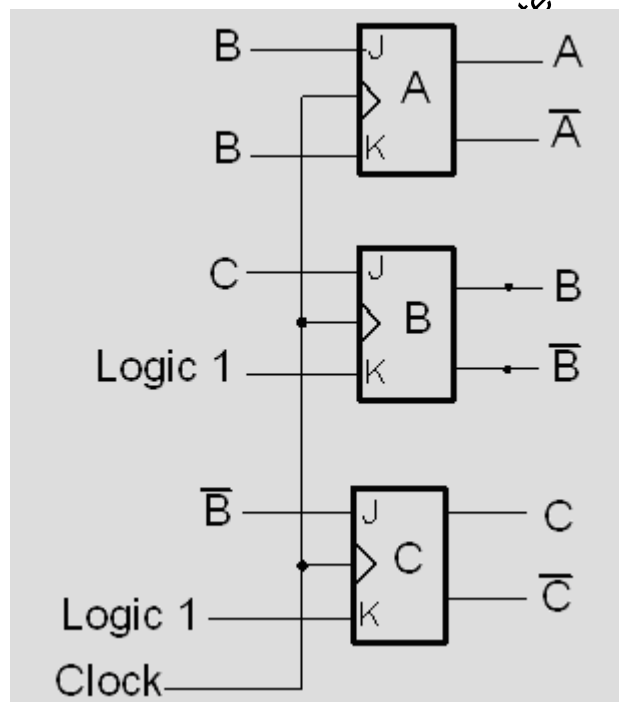
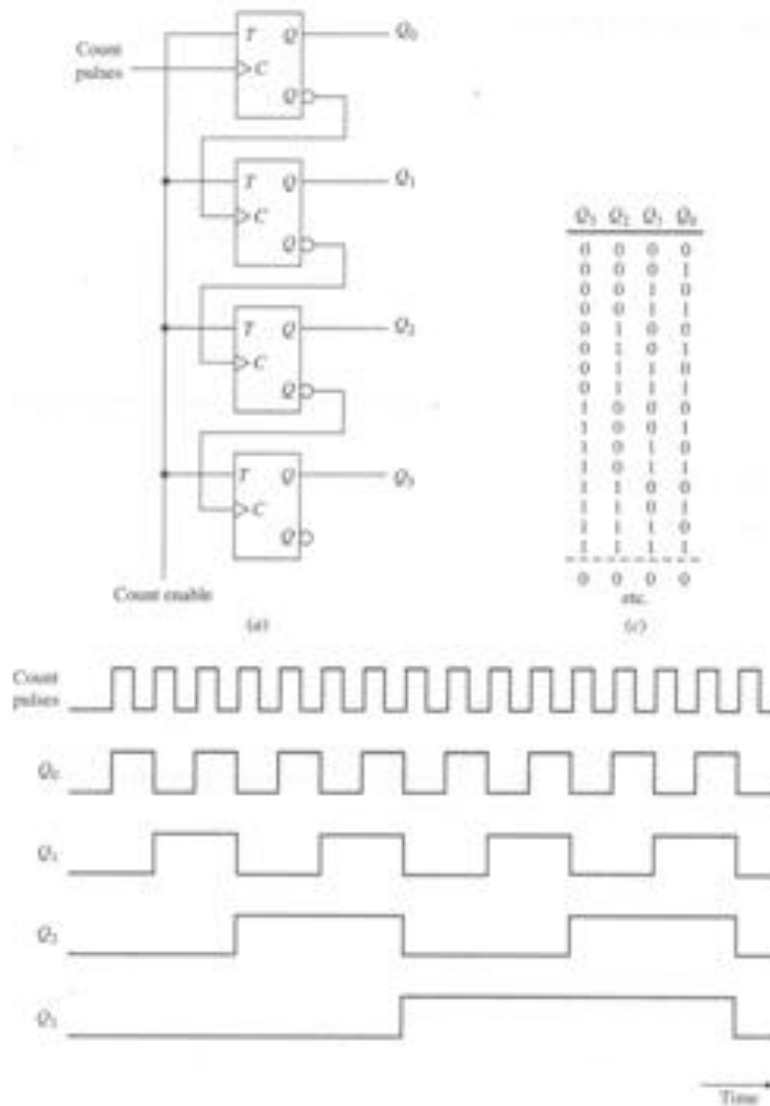


Figure 4: Circuit for arbitrary counting sequence

**4-bit Binary Ripple Counter :**



- All Flip-Flops are in toggle mode.
- The clock input is applied.
- Count enable = 1.
- Counter counts from 0000 to 1111.

## Counters

In this lesson, the operation and design of Synchronous Binary Counters will be studied.

### Synchronous Binary Counters (SBC)

#### Description and Operation

In its simplest form, a *synchronous binary counter (SBC)* receives a train of clock *pulses* as input and outputs the *pulse count* ( $Q_{n-1} \dots Q_2 Q_1 Q_0$ ).

An example is a 3-bit counter that counts from 000 upto 111. Each counter consists of a number of FFs. (Figure 1)

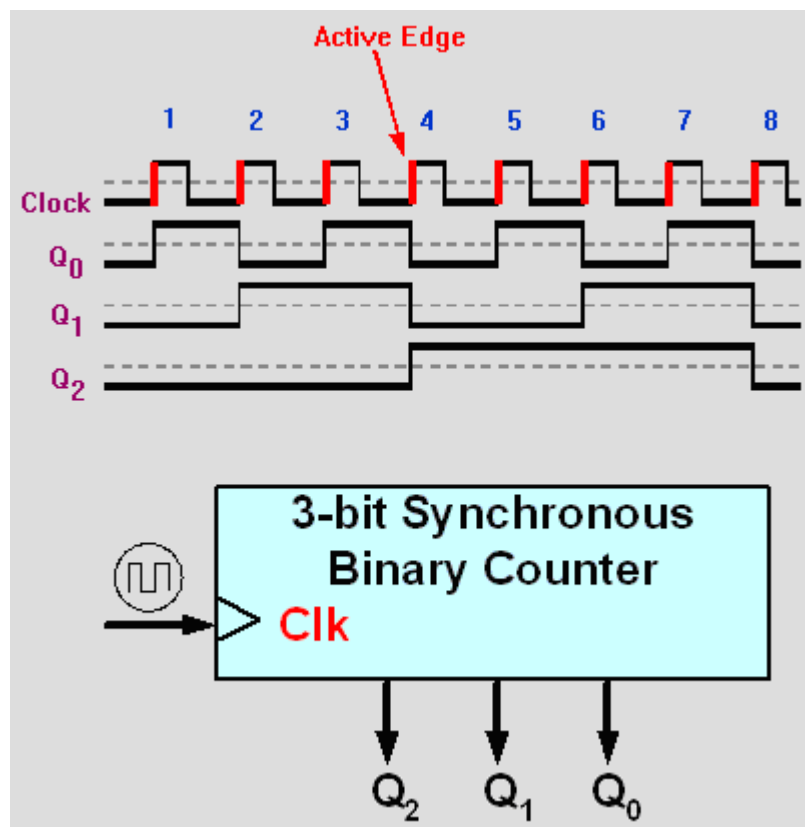


Figure 1: 3-bit SBC

In *synchronous* counters, all FFs are triggered by the same input clock.

An  $n$ -bit counter has  $n$ -FFs with  $2^n$  distinct *states*, where each state corresponds to a particular *count*.

Accordingly, the possible *counts* of an  $n$ -bit counter are 0 to  $(2^n - 1)$ . Moreover an  $n$ -bit

counter has  $n$  output bits ( $Q_{n-1} \dots Q_2 Q_1 Q_0$ ).

After reaching the maximum count of  $(2^n - 1)$ , the following clock pulse resets the count back to 0.

Thus, a 3-bit counter counts from 0 to 7 and back to 0. In other words, the output count actually equals *(Total # of input pulses Modulo  $2^n$ )*.

Accordingly, it is common to identify counters by the modulus  $2^n$ . For example, a 4-bit counter provides a modulo 16 count, a 3-bit counter is a modulo 8 counter, etc.

Referring to the 3-bit counter mentioned earlier, each stage of the counter divides the frequency by 2, where the last stage divides the frequency by  $2^n$ ,  $n$  being the number of bits. (Figure 2)

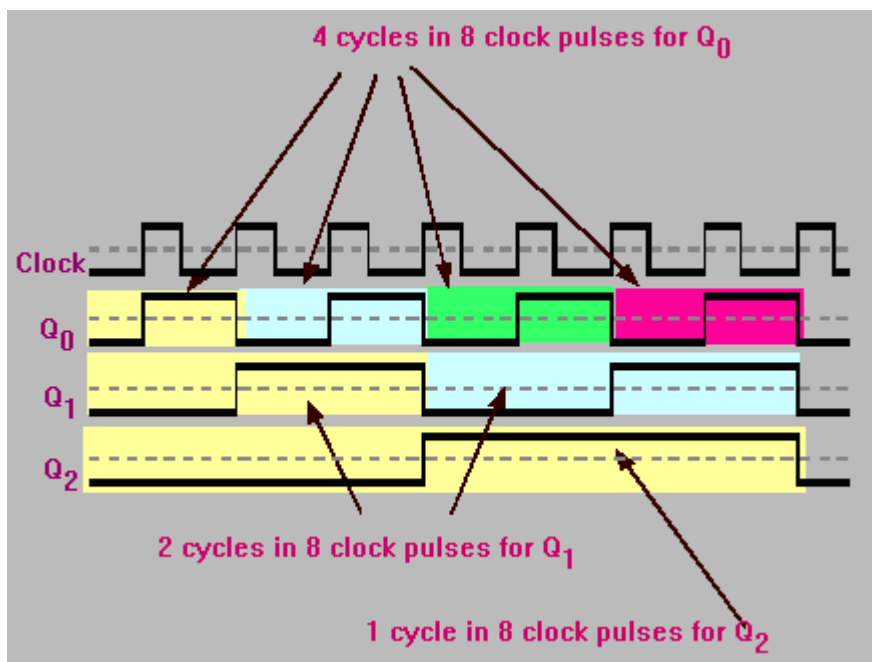


Figure 2: 3-bit SBC

Thus, if the frequency (i.e. no. of cycles/ sec) of clock is  $F$ , then the frequency of output waveform of  $Q_0$  is  $F/2$ ,  $Q_1$  is  $F/4$ , and so on. In general, for  $n$ -bit counter, we have  $F/2^n$ .

### Design of Binary Counters (SBC)

Design procedure is the same as for other synchronous circuits.

A counter may operate without an external input (except for the clock pulses!)

In this case, the output of the counter is taken from the outputs of the flip-flops without



any additional outputs from gates.

Thus, there are no columns for the input and outputs in the state table; we only see the current state and next state...

### Example Design a 4-bit SBC using JK flip-flops.

The counter has 4 FFs with a total of 16 states, (0000 to 1111) \_ 4 state variables Q3 Q2 Q1 Q0 are required.

Present State				Next State				Flip-Flop Inputs							
Q <sub>3</sub>	Q <sub>2</sub>	Q <sub>1</sub>	Q <sub>0</sub>	Q <sub>3</sub>	Q <sub>2</sub>	Q <sub>1</sub>	Q <sub>0</sub>	J <sub>Q3</sub>	K <sub>Q3</sub>	J <sub>Q2</sub>	K <sub>Q2</sub>	J <sub>Q1</sub>	K <sub>Q1</sub>	J <sub>Q0</sub>	K <sub>Q0</sub>
0	0	0	0	0	0	0	1	0	X	0	X	0	X	1	X
0	0	0	1	0	0	1	0	0	X	0	X	1	X	X	1
0	0	1	0	0	0	1	1	0	X	0	X	X	0	1	X
0	0	1	1	0	1	0	0	0	X	1	X	X	1	X	1
0	1	0	0	0	1	0	1	0	X	X	0	0	X	1	X
0	1	0	1	0	1	1	0	0	X	X	0	1	X	X	1
0	1	1	0	0	1	1	1	0	X	X	0	X	0	1	X
0	1	1	1	1	0	0	0	1	X	X	1	X	1	X	1
1	0	0	0	1	0	0	1	X	0	0	X	0	X	1	X
1	0	0	1	1	0	1	0	X	0	0	X	1	X	X	1
1	0	1	0	1	0	1	1	X	0	0	X	X	0	1	X
1	0	1	1	1	1	0	0	X	0	1	X	X	1	X	1
1	1	0	0	1	1	0	1	X	0	X	0	0	X	1	X
1	1	0	1	1	1	1	0	X	0	X	0	1	X	X	1
1	1	1	0	1	1	1	1	X	0	X	0	X	0	1	X
1	1	1	1	0	0	0	0	X	1	X	1	X	1	X	1

Figure 3: State table for the example

Notice that the next state equals the present state plus one.

To design this circuit, we derive the flip-flop input equations from the state transition table. Recall that to find J & K values, we have to use:

- \_ The present state,
- \_ The next state, and
- \_ The JK flip-flop excitation table.

When the *count* reaches 1111, it resets back to 0000, and the count cycle is repeated.

Once the J and K values are obtained, the next step is to find out the simplified input equations by using K-maps, as shown in figure 4.

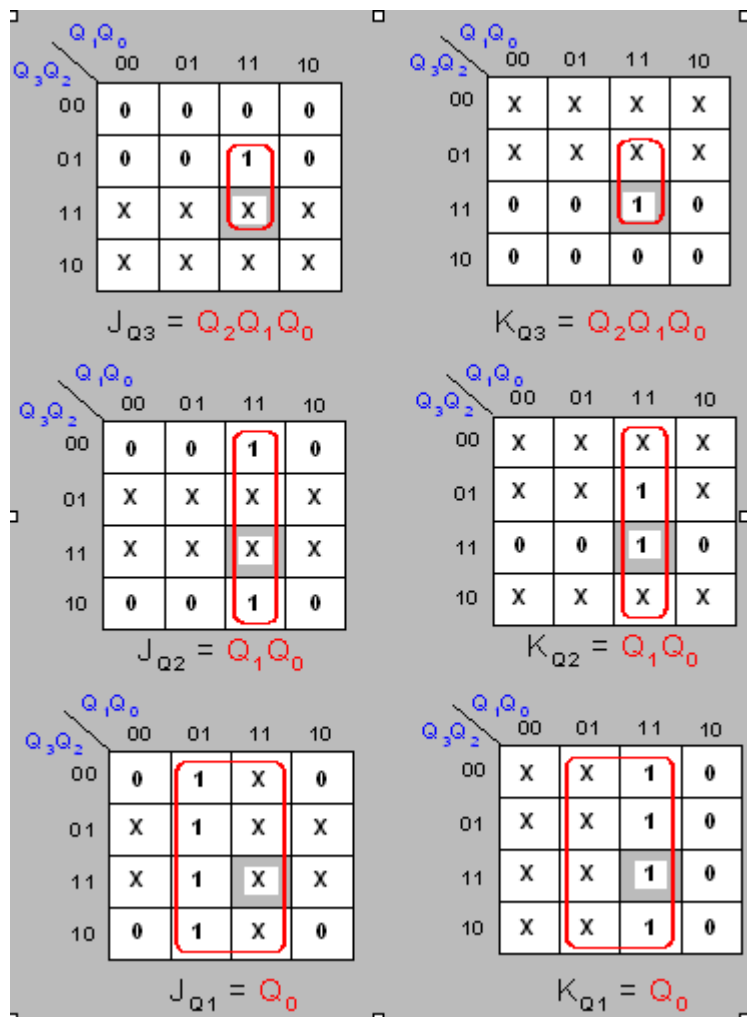


Figure 4: K-maps for the example

Notice that the maps for  $J_{Q0}$  and  $K_{Q0}$  are not drawn because the values in the table for these two variables either contain 1's or X's. This will result in  $J_{Q0} = K_{Q0} = 1$

Note that the Boolean equation for J input is the same as that of the K input for all the FFs  $\Rightarrow$  Can use T-FFs instead of JK-FFs.

### Count Enable Control

In many applications, controlling the counting operation is necessary  $\Rightarrow$  a count-enable (En) is required.

**If** En= 1 **then** counting of incoming clock pulses is enabled **Else if** (En =0), no incoming clock pulse is counted.

To accommodate the enable control, two approaches are possible.

1. Controlling the clock input of the counter
2. Controlling FF excitation inputs (JK, T, D, etc.).

### Clock Control

Here, instead of applying the system clock to the counter directly, the clock is first ANDed with the En signal.

Even though this approach is simple, it is not recommended to use particularly with configurable logic, e.g. FPGA's.

### FF Input Control (Figure 5)

In this case, the  $En = 0$  causes the FF inputs to assume the no change value ( $SR=00$ ,  $JK=00$ ,  $T=0$ , or  $Di=Qi$ ).

To include En, analyze the stage when  $JQ1 = KQ1 = Q0$ , and then include En. Accordingly, the FF input equations of the previous 4-bit counter example will be modified as follows:

$$JQ0 = KQ0 = 1. En = En$$

$$JQ1 = KQ1 = Q0. En$$

$$JQ2 = KQ2 = Q1.Q0. En$$

$$JQ3 = KQ3 = Q2.Q1.Q0. En$$

notes4free.in

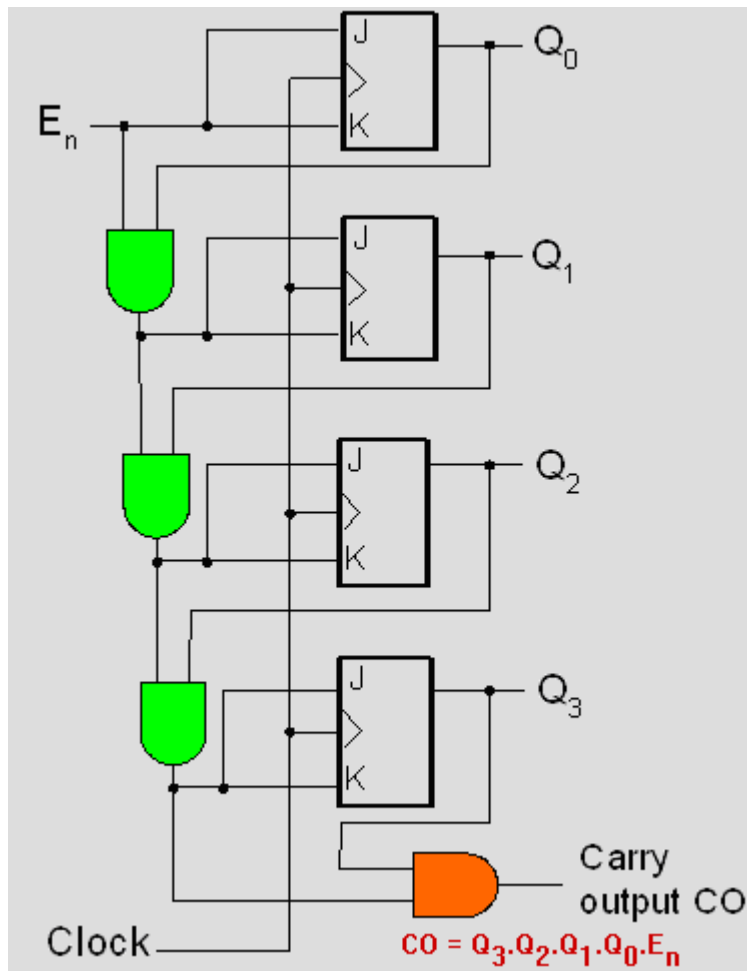


Figure 5: FF input control in counter

Thus, when  $E_n = 0$ , all J and K inputs are equal to zero, and the flip flops remain in the same state, even in the presence of clock pulses

When  $E_n = 1$ , the input equations are the same as equations of the previous example.

A carry output signal (CO) is generated when the counting cycle is complete, as seen in the timing diagram.

The CO can be used to allow cascading of two counters while using the same clock for both counters. In that case, the CO from the first counter becomes the  $E_n$  for the second counter. For example, two modulo-16 counters can be cascaded to form a modulo-256 counter.

Up-Down Binary Counters

In addition to counting up, a SBC can be made to count down as well.

A control input,  $S$  is required to control the direction of count.

IF  $S=1$ , the counter counts up, otherwise it counts down.

### FF Input Control

Design a Modulo-8 up-down counter with control input  $S$ , such that if  $S=1$ , the counter counts up, otherwise it counts down. Show how to provide a count enable input and a carry-out (CO) output. (See figures 6 & 7)

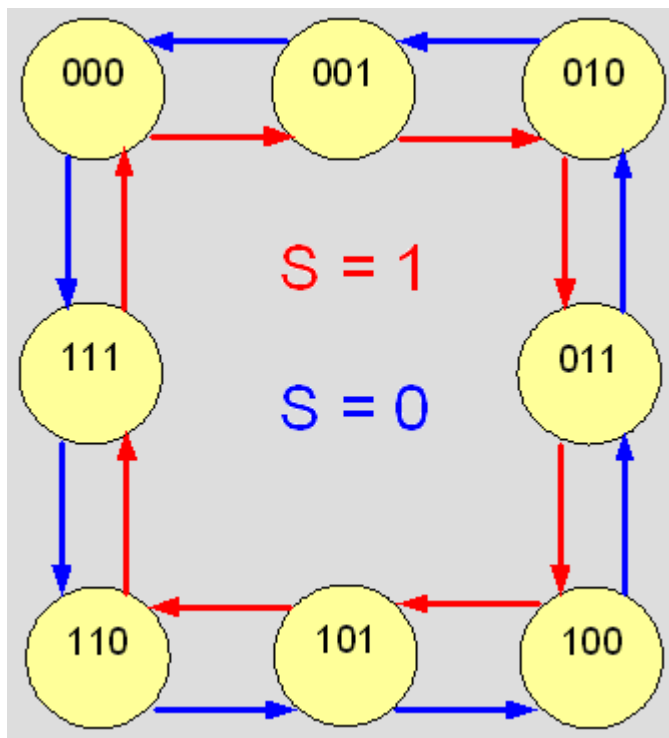


Figure 6: State diagram for FF input control example

Present State				Next State			Flip-Flop Inputs		
Q <sub>2</sub>	Q <sub>1</sub>	Q <sub>0</sub>	S	Q <sub>2</sub>	Q <sub>1</sub>	Q <sub>0</sub>	T <sub>2</sub>	T <sub>1</sub>	T <sub>0</sub>
0	0	0	0	1	1	1	1	1	1
0	0	0	1	0	0	1	0	0	1
0	0	1	0	0	0	0	0	0	1
0	0	1	1	0	1	0	0	1	1
0	1	0	0	0	0	1	0	1	1
0	1	0	1	0	1	1	0	0	1
0	1	1	0	0	1	0	0	0	1
0	1	1	1	1	0	0	1	1	1
1	0	0	0	0	1	1	1	1	1
1	0	0	1	1	0	1	0	0	1
1	0	1	0	1	0	0	0	0	1
1	0	1	1	1	1	0	0	1	1
1	1	0	0	1	0	1	0	1	1
1	1	0	1	1	1	1	0	0	1
1	1	1	0	1	1	0	0	0	1
1	1	1	1	0	0	0	1	1	1

Figure 7: State table for FF input control example

The equations are (see figure 8)

$$T_0 = 1$$

$$T_1 = Q_0 \cdot S + Q_0 \cdot \bar{S}$$

$$0 \cdot S + \bar{Q}_0$$

$$T_2 = Q_1 \cdot Q_0 \cdot S + Q_1 \cdot Q_0 \cdot \bar{S}$$

$$1 \cdot Q_1 \cdot Q_0$$

$$0 \cdot S + \bar{Q}_1 \cdot \bar{Q}_0$$

The carry outputs for the next stage are: (see figure 8)

$$C_{up} = Q_2 \cdot Q_1 \cdot Q_0 \text{ for upward counting.}$$

$$C_{down} = Q_2 \cdot \bar{Q}_1 \cdot \bar{Q}_0$$

$$2 \cdot Q_2 \cdot \bar{Q}_1 \cdot \bar{Q}_0$$

$$1 \cdot Q_2 \cdot \bar{Q}_1 \cdot \bar{Q}_0$$

0 for downward counting.

The equations with  $E_n$  are (see figure 9)

$$T_0 = E_n \cdot 1$$

$$T1 = Q0 \cdot S \cdot En + Q0$$

$$0 \cdot S \cdot En$$

$$T2 = Q1 \cdot Q0 \cdot S \cdot En + Q1$$

$$1 \cdot Q1$$

$$0 \cdot S \cdot En$$

The carry outputs for the next stage, with En are (see figure 9):

$$Cup = Q2 \cdot Q1 \cdot Q0 \cdot En \text{ for counting up.}$$

$$Cdown = Q1 \cdot Q0$$

$$2 \cdot Q1$$

$$1 \cdot Q1$$

$$0 \cdot En \text{ for counting down.}$$

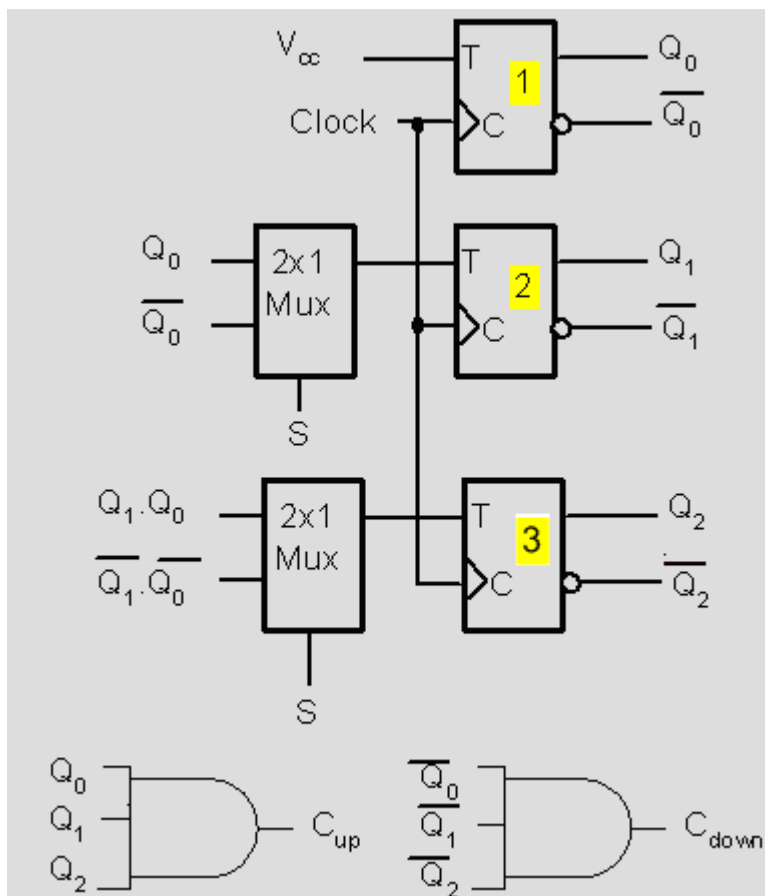


Figure 8: Circuit of up-down counter

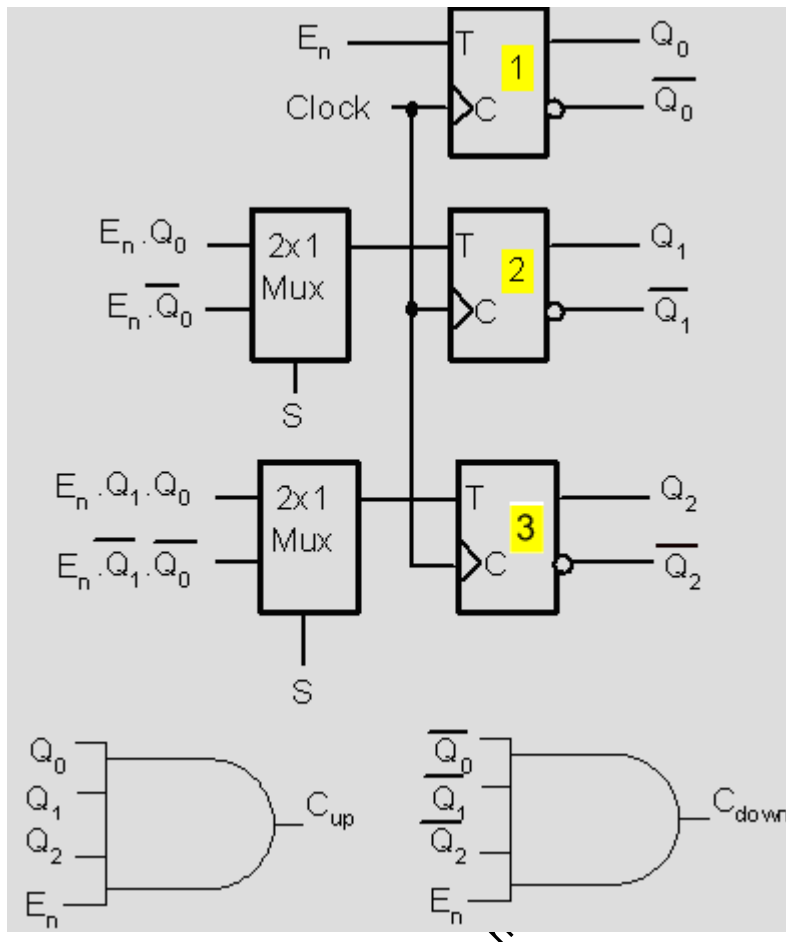
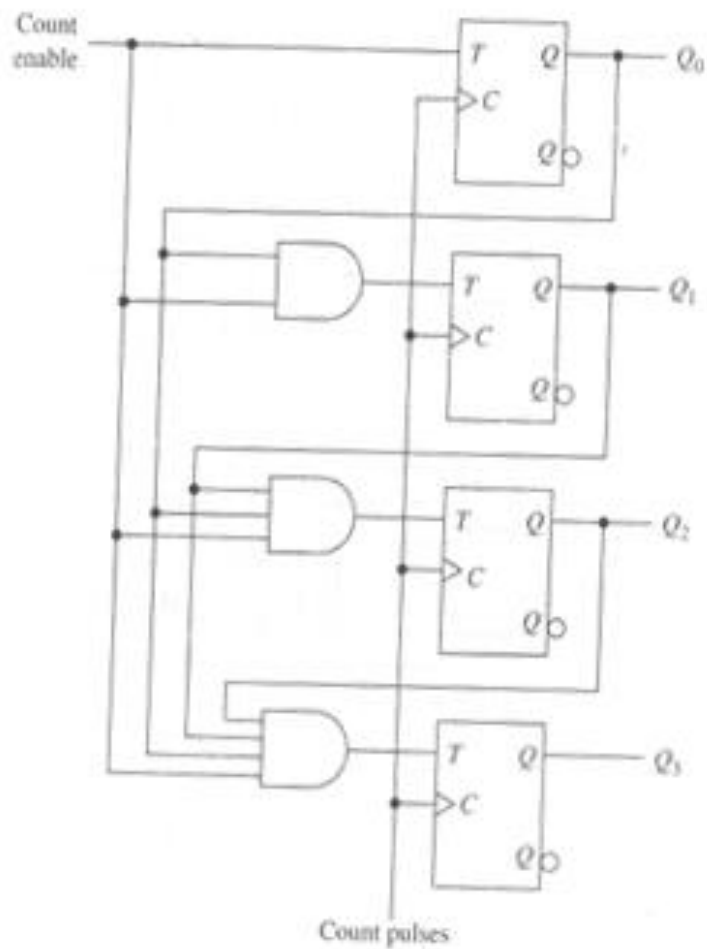


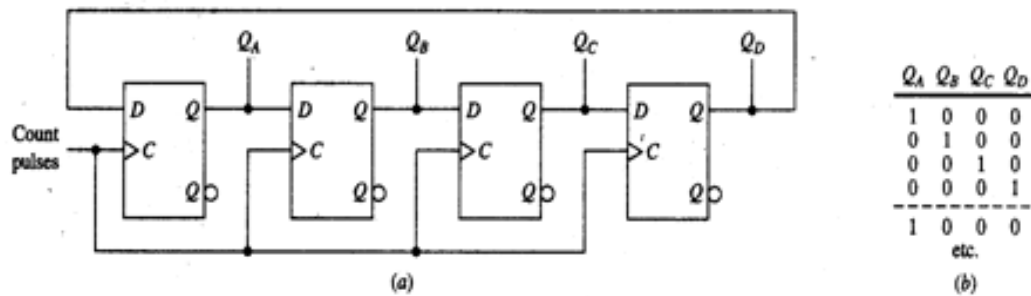
Figure 9: Circuit of up-down counter with  $E_n$



**Synchronous Binary Counter :**

- The clock input is common to all Flip-Flops.
- The T input is function of the output of previous flip-flop.
- Extra combination circuit is required for flip-flop input.

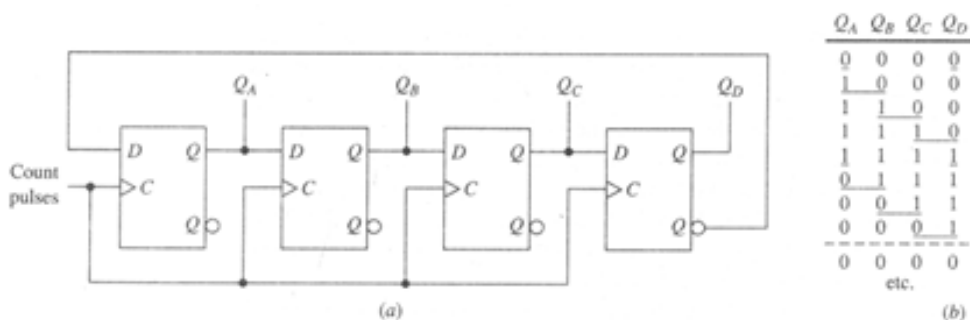
Counters Based on Shift Register



**Mod-4 Ring Counter**

- The output of LSB FF is connected as D input to MSB FF.
- This is commonly called as Ring Counter or Circular Counter.
- The data is shifted to right with each clock pulse.
- This counter has four different states.
- This can be extended to any no. of bits.

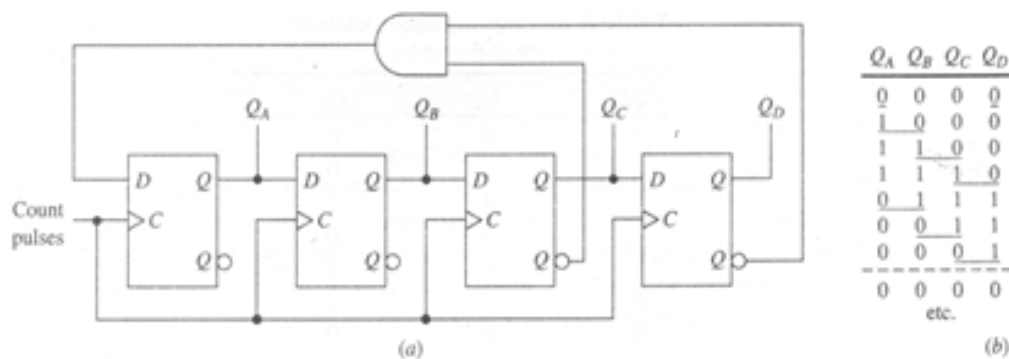
Twisted Ring Counter or Johnson Counter



**Mod-8 Johnson Counter**

- The complement output of LSB FF is connected as D input to MSB FF.
- This is commonly called as Johnson Counter.
- The data is shifted to right with each clock pulse.
- This counter has eight different states.
- This can be extended to any no. of bits.

### Mod-7 Twisted Ring Counter



### Mod-7 Ring Counter

- The D input to MSB FF is  $\overline{Q_D} \cdot \overline{Q_C}$
- The counter follows seven different states with application of clock input.
- By changing feedback different counters can be obtained.

### Design Procedure for Synchronous Counter

- The clock input is common to all Flip-Flops.
- Any Flip-Flop can be used.
- For mod-n counter 0 to n-1 are counter states.
- The excitation table is written considering the present state and next state of counter.
- The flip-flop inputs are obtained from characteristic equation.
- By using flip-flops and logic gate the implementation of synchronous counter is obtained.

**Difference between Asynchronous and Synchronous Counter :**

Asynchronous Counter	Synchronous Counter
1. Clock input is applied to LSB FF. The output of first FF is connected as clock to next FF.	1. Clock input is common to all FF.
2. All Flip-Flops are toggle FF.	2. Any FF can be used.
3. Speed depends on no. of FF used for n bit . $f_{max} = \frac{1}{n \times t_p}$	3. Speed is independent of no. of FF used. $f_{max} = \frac{1}{t_p}$
4. No extra Logic Gates are required.	4. Logic Gates are required based on design.
5. Cost is less.	5. Cost is more.

notes4free.in

**Recommended question and answer –unit-6**

**Jan -2009**

b) Explain the working principle of a mod-6 binary ripple counter, configured using positive edge triggered T-FF. Also draw the timing diagram.

(8),

Ans. : Mod-8 ripple counter using T flip flop: For designing counter using T flip flop,

Flip-flops required are :  $2^n \geq N$

Here  $N = 8 \therefore n = 3$

i.e. 3 FFs are required

Excitation table for T FF :

$Q_n$	$Q_{n+1}$	T
0	0	0
0	1	1
1	0	1
1	1	0

Transition table :

Present state			Next state			Flip-flop inputs		
A	B	C	$A^+$	$B^+$	$C^+$	$T_A$	$T_B$	$T_C$
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	1
0	1	0	0	1	1	0	0	1
0	1	1	1	0	0	1	1	1
1	0	0	1	0	1	0	0	1
1	0	1	1	1	0	0	1	1
1	1	0	1	1	1	0	0	1
1	1	1	0	0	0	1	1	1

K-map simplification :

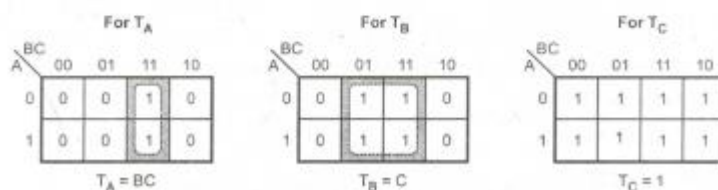


Fig. 6 (a)

Implement the counter :

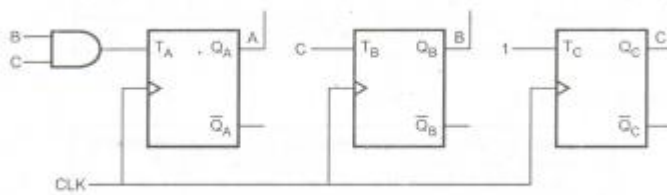


Fig. 6 (b)

**Jan-2008**

i) Synchronous and asynchronous circuits.

ii) Combinational and sequential circuits.

Ans. : i) Synchronous and asynchronous circuits :

Sr. No.	Synchronous sequential circuits	Asynchronous sequential circuits
1.	In synchronous circuits, memory elements are clocked flip-flops.	In asynchronous circuits, memory elements are either unlocked flip-flops or time delay elements.
2.	In synchronous circuits, the change in input signals can affect memory element upon activation of clock signal.	In asynchronous circuits change in input signals can affect memory element at any instant of time.
3.	The maximum operating speed of clock depends on time delays involved.	Because of absence of clock, asynchronous circuits can operate faster than synchronous circuits.
4.	Easier to design.	More difficult to design.

b) Explain the working of 4-bit asynchronous counter.

Ans. : 4-bit asynchronous counter :

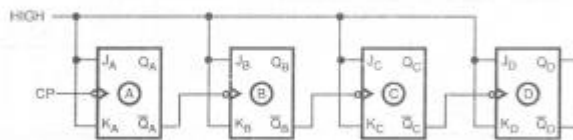


Fig. 10 4-bit asynchronous down counter

- 1) 4 flip-flops are employed to create a 4-bit asynchronous counter as shown.
  - 2) The clock signal is connected to the clock input of only first stage flip-flop.
  - 3) Because of the inherent propagation delay time through a flip-flop, two flip-flops never trigger simultaneously. Thus, it works in an asynchronous operation.
  - 4) Output of the first flip-flop triggers the second flip-flop and so on.
- S) At the output of flip-flops, we get the counted value of the counter.

c) Explain Johnson counter with its circuit diagram and timing diagram. (8)

Ans. : 4-bit Johnson counter :

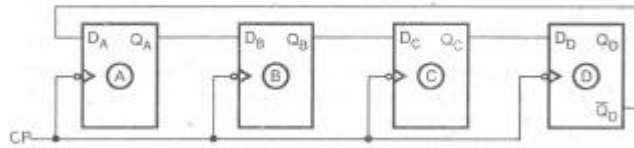


Fig. 11 Four-bit Johnson counter

1) Initially, the register is cleared.

∴ all the outputs QA, QB, QC, QD are zero.

2) The complement of Q0 is 1 which is connected back to the D input of first stage.

∴ DA is, 1.

∴ The output becomes QA = 1, QB = 0, QC = 0 and QD = 0.

3) The next clock pulse produces QA = 1, QB = 1, QC = 0 and QD = 0.

The sequence is given as :

Clock Pulse	QA	QB	QC	QD
0	0	0	0	0
1	1	0	0	0
2	1	1	0	0
3	1	1	1	0
4	1	1	1	1
5	0	1	1	1
6	0	0	1	1
7	0	0	0	1

Table 3 Four-bit Johnson sequence

Aug-2008

Q.5 a) Derive the characteristics equations of the following flip-flops.

- i) SR flip-flops
- ii) JK flip-flop

(10)

Ans. : i) Function table of SR flip-flop

S	R	Q	Ck	Q+
0	0	0	-	0
0	0	1	-	1
0	1	0	-	0
0	1	1	-	0
1	0	0	-	1
1	0	1	-	1
1	1	0	-	X
1	1	1	-	X
X	X	Q	0	Q
X	X	Q	1	Q

Characteristic equation :

	SR	00	01	11	10
Q	0	0	0	X	1
	1	1	0	X	1

$$Q^+ = S + \bar{R}Q$$

Fig. 8 (a)

In words :

- If SR = 10, on high going edge of clock the Q output becomes 1.
- If SR = 01, Q becomes 0.
- If SR = 11, Q raised condition.
- If SR = 00, Q does not change.

ii) Function table of JK flip-flop

J	K	Q	Ck	Q+
0	0	0	-	0
0	0	1	-	1
0	1	0	-	0
0	1	1	-	0
1	0	0	-	1
1	0	1	-	1
1	1	0	-	1
1	1	1	-	0
X	X	Q	0	Q
X	X	Q	1	Q

Characteristic equation

	JK	00	01	11	10
Q	0	0	0	1	1
	1	1	0	0	0

$$Q^+ = \bar{Q}J + Q\bar{K}$$

Fig. 8(b)

In words :

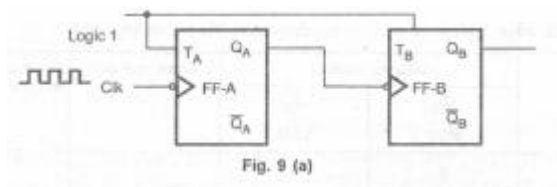


- If JK = 10, On high going edge of clock, the Q output becomes 1.
- If JK = 01, Q becomes 0.
- If JK = 11, Q toggles.
- If JK = 00, Q does not change.

b) Explain clearly the operation of an asynchronous inputs in a flip-flops with suitable example. (6)

Ans. : The best example of operation of asynchronous input to flip-flop is counter.

Use of counter is to count the clock pulse. For these counters the external clock signal is applied to one flip-flop and then the output of preceding flip-flop is connected to the clock of next flip-flop



Operation:

1) Initially both the flip-flops be in reset condition.

..  $Q_B Q_A = 00$

2) On the first negative going clock edge : As the 1st falling edge of the clock hits FF-A, it will toggle as  $T_A = 1$ . Hence  $Q_A$  will be equal to 1. But for FF - B it has not changed from 0 to 1, it is treated as the positive clock edge by FF - B.

So

$Q_B Q_A = 01$  ... After the first clk pulse

On second falling edge of clock pulse :

On arrival of second falling clock edge, FF-A toggles again, to make  $Q_A = 0$ . This change in  $Q_A$  (from 1 to 0) acts as a negative clock edge for FF-B. So it will also toggle, and  $Q_B$  will become 1.

Hence after the second clock pulse the counter output are

$Q_B Q_A = 10$  ... After the second clk pulse

Both the outputs are changing their state. But both the changes do not take place

simultaneously, QA will change first from 1 to 0 and then QB will change from 0 to 1. This is due to propagation delay of FF-A. So both flip-flops will never triggered at the same instant. So it is asynchronous counter

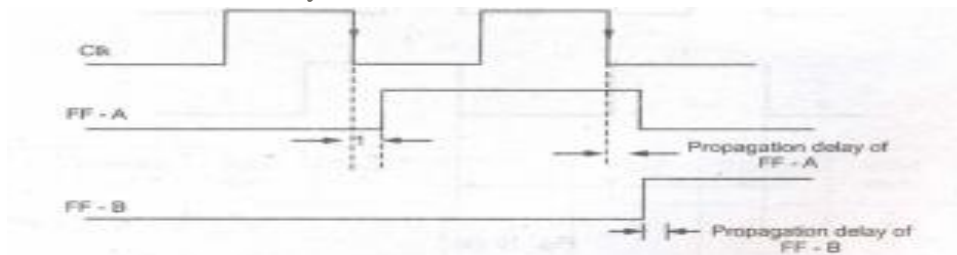


Fig. 9 (b)

For 4 clock edge operation will be as shown in the following table.

Clock	Counter output		State number	Decimal equation of counter output
	QB MSB	QA LSB		
Initially	0	0	-	0
1 ↓	0	1	1	1
2 ↓	1	0	2	2
3 ↓	1	1	3	3
4 ↓	0	0	4	0

c) An edge triggered 'D' flip-flop is connected as shown in the Fig. 10. Assume that An Q = 0 initially and sketch the waveform and determine its frequency of the signal at 'Q' output. (4)

Ans. :

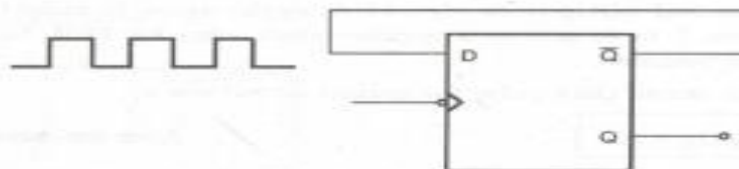


Fig. 10

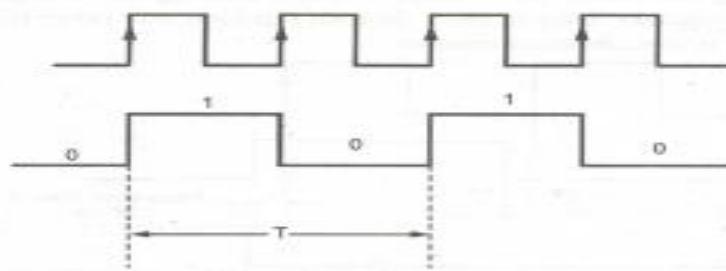


Fig. 10 (a)

$$f = \frac{1}{T} = f = \frac{f_{\text{clock}}}{2}$$

Aug-2007

c) Write the truth table of the following flip flops :

D, T, SR, JK.

Sol. : D :

CP	D	$Q_{n+1}$
↓	0	0
↓	1	1
0	X	$Q_n$

Truth table of D flip-flop

Table 2

Logic Diagram

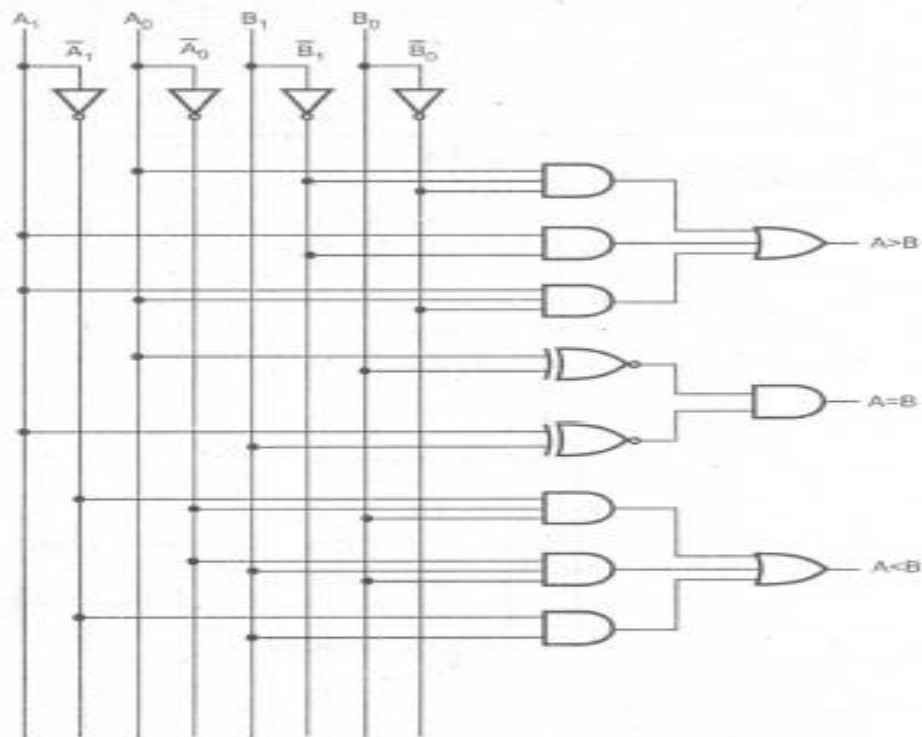


Fig. 9

b) Implement the following Boolean function using 8:1 MUX :

$$F(A, B, C, D) = \Sigma m(1, 2, 5, 9, 10, 14)$$

Sol. : Implementation Table :

T :

$Q_n$	T	$Q_{n+1}$
0	0	0
0	1	1
1	0	1
1	1	0

 $\equiv$ 

T	$Q_{n+1}$
0	$Q_n$
1	$\overline{Q_n}$

(b) Truth table (3)

SR :

EN	S	R	$Q_n$	$Q_{n+1}$	State
1	0	0	0	0	No change (NC)
1	0	0	1	1	
1	0	1	0	0	Reset
1	0	1	1	0	
1	1	0	0	1	Set
1	1	0	1	1	
1	1	1	0	X	Indeterminate
1	1	1	1	X	
0	X	X	0	0	No change (NC)
0	X	X	1	1	

Table 4 Truth table for SR latch with enable input

JK :

J	K	$Q_{n+1}$
0	0	$Q_n$
0	1	0
1	0	1
1	1	$\overline{Q_n}$

Truth table 5

notes4fre

Q.7 a) Realize a 3 bit binary synchronous up counter using JK flip flop. Write the-excitation table, transition table and logic diagram. Include preset, clear option. [10]

Sol : 3 - bit binary synchronous up counter using JK flip-flops :

Fig. 12 (a) shows 3-bit synchronous binary counter and its timing diagram. The state sequence for this counter is shown in Table 6.

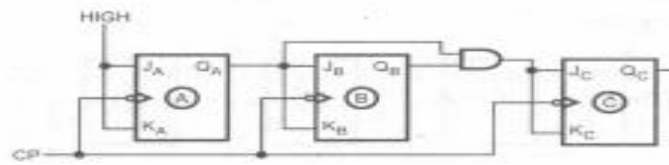


Fig. 12 (a) A three-bit synchronous binary counter



Fig. 12 (b) Timing diagram for 3-bit synchronous binary counter

CP	QC	QB	QA
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

Table 6 State sequence for 3-bit binary counter

Looking at Fig. 12 (b), we can see that QA changes on each clock pulse as we progress from its original state to its final state and then back to its original state. To produce this operation, flip-flop A is held in the toggle mode by connecting J and K inputs to HIGH. Now let us see what flip-flop B does. Flip-flop B toggles, when QA is 1. When QA is a 0, flip-flop B is in the no-change mode and remains in its present state. Looking at the Table 6 we can notice that flip-flop C has to change its state only when QB and QA both are at logic 1. This condition is detected by AND gate and

**MODULE-5**

**Hours-10**

**Sequential Design - I:**

Introduction, Mealy and Moore Models, State Machine Notation, Synchronous Sequential Circuit Analysis

**Recommended readings:**

1. Donald D Givone, "Digital Principles and Design ", Tata McGraw Hill Edition, 2002.

Units-6.1, 6.2, 6.3

notes4free.in

## Mealy and Moore Type Finite State Machines

### Objectives

There are two basic ways to design clocked sequential circuits. These are using:

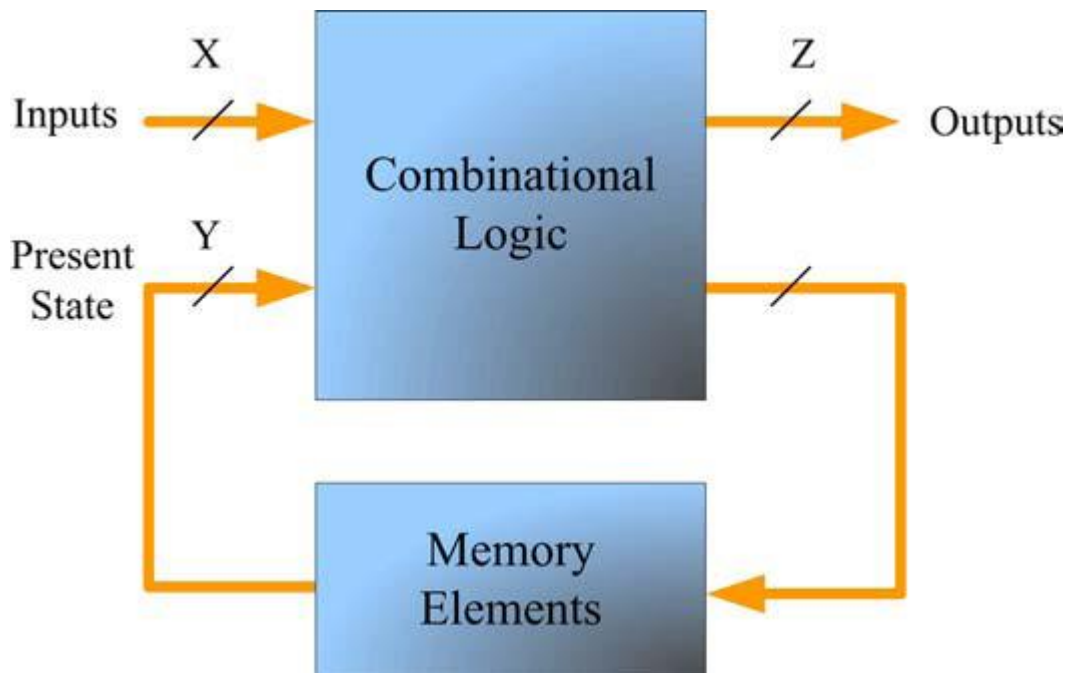
1. Mealy Machine, which we have seen so far.
2. Moore Machine.

The objectives of this lesson are:

1. Study Mealy and Moore machines
2. Comparison of the two machine types
3. Timing diagram and state machines

### Mealy Machine

- In a Mealy machine, the outputs are a function of the present state and the value of the inputs as shown in Figure 1.
- Accordingly, the outputs may change asynchronously in response to any change in the inputs.



**Figure 1: Mealy Type Machine**

### Mealy Machine

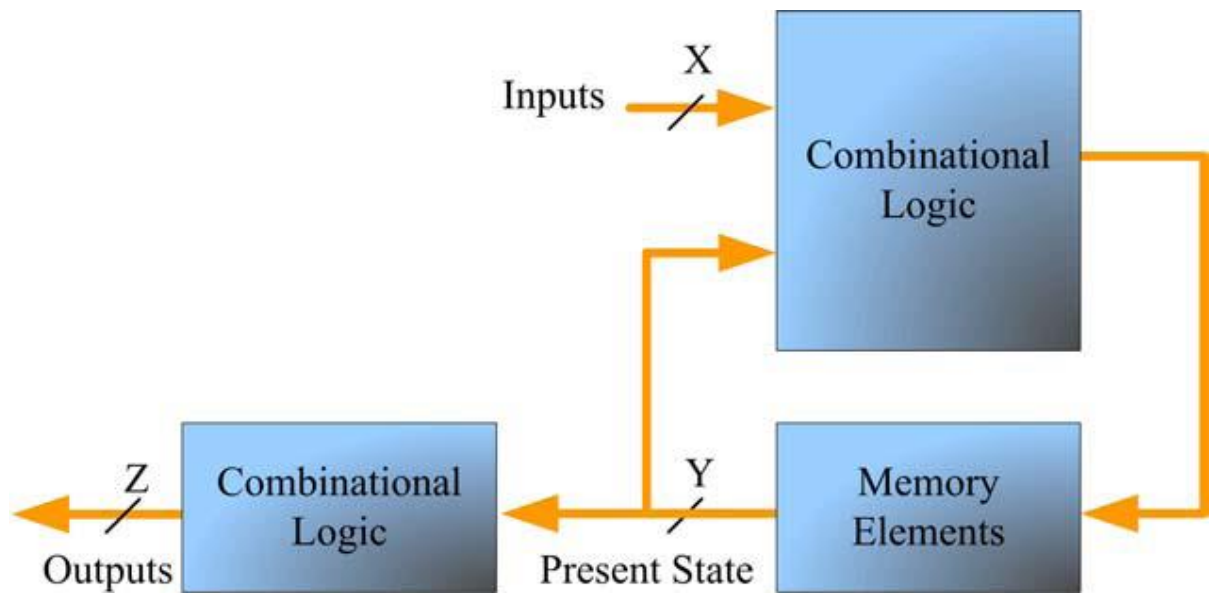
In a Moore machine the outputs depend only on the present state as shown in Figure 2.

A combinational logic block maps the inputs and the current state into the necessary flip-flop inputs to store the appropriate next state just like Mealy machine.

However, the outputs are computed by a combinational logic block whose inputs are only the flip-flops state outputs.

The outputs change synchronously with the state transition triggered by the active clock edge.





**Figure 2: Moore Type Machine**

### Comparison of the Two Machine Types

notes4free.in

Consider a finite state machine that checks for a pattern of '10' and asserts logic high when it is detected.

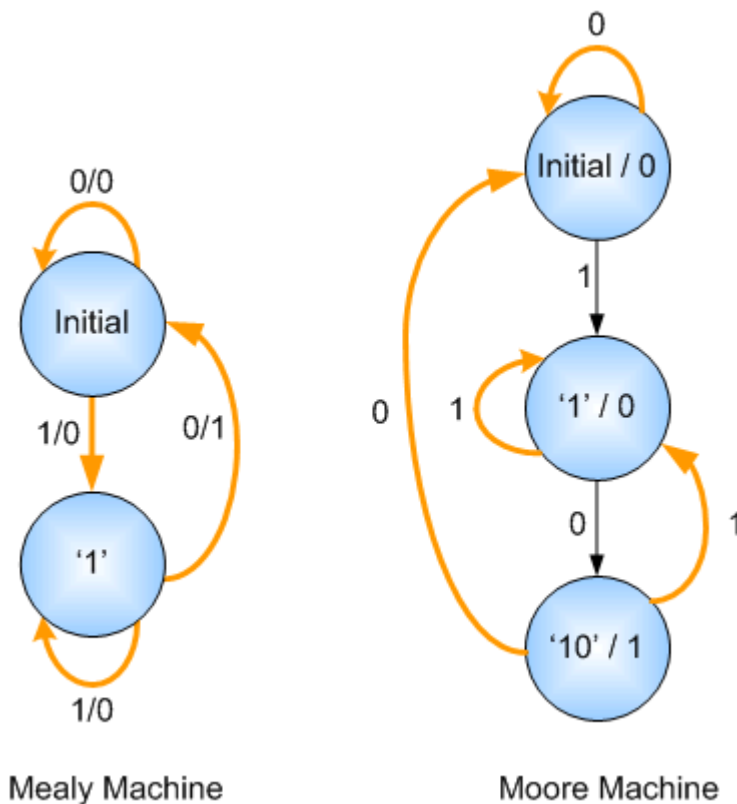
The state diagram representations for the Mealy and Moore machines are shown in Figure 3.

The state diagram of the Mealy machine lists the inputs with their associated outputs on state transitions arcs.

The value stated on the arrows for Mealy machine is of the form  $Z_i/X_i$  where  $Z_i$  represents input value and  $X_i$  represents output value.

A Moore machine produces a unique output for every state irrespective of inputs.

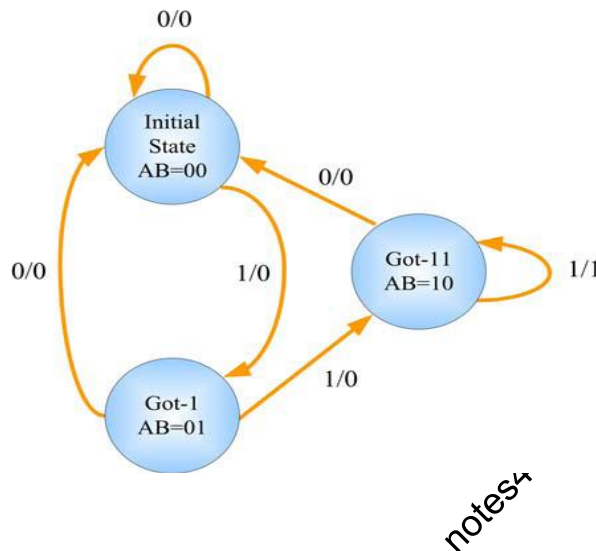
- Accordingly the state diagram of the Moore machine associates the output with the state in the form state-notation/output-value.
- The state transition arrows of Moore machine are labeled with the input value that triggers such transition.
- Since a Mealy machine associates outputs with transitions, an output sequence can be generated in fewer states using Mealy machine as compared to Moore machine. This was illustrated in the previous example.



**Figure 3: Mealy and Moore State Diagrams for '10' Sequence Detector**

### Timing Diagrams

- □ To analyze Mealy and Moore machine timings, consider the following problem. A state-machine outputs '1' if the input is '1' for three consecutive clocks.



**Figure 4: Mealy State Machine for '111' Sequence Detector**

### Mealy State Machine

- The Mealy machine state diagram is shown in Figure 4.
- Note that there is no reset condition in the state machine that employs two flip-flops. This means that the state machine can enter its unused state '11' on start up.
- To make sure that machine gets resetted to a valid state, we use a 'Reset' signal.

- The logic diagram for this state machine is shown in Figure 5. Note that negative edge triggered flip-flops are used.

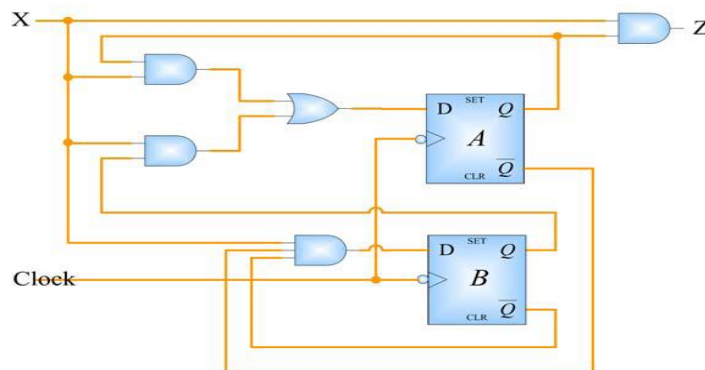
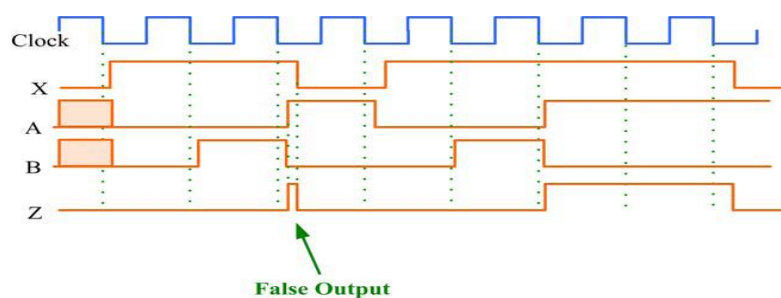


Figure 5: Mealy State Machine Circuit Implementation

notes4free.in

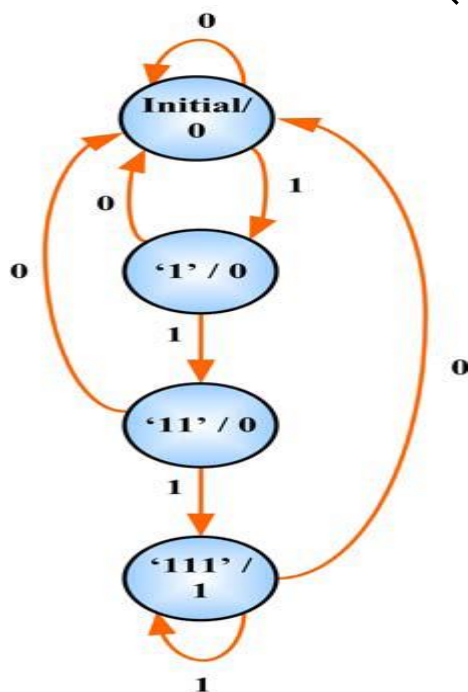
- Since the output in Mealy model is a combination of present state and input values, an unsynchronized input with triggering clock may result in invalid output, as in the present case.
- Consider the present case where input 'x' remains high for sometime after state 'AB = 10' is reached. This results in 'False Output', also known as 'Output Glitch'.



**Figure 6: Timing Diagram for Mealy Model Sequence Detector**

### Moore State Machine

- The Moore machine state diagram for '111' sequence detector is shown in Figure 7.
- The state diagram is converted into its equivalent state table (See Table 1).
- The states are next encoded with binary values and we achieve a state transition table (See Table 2).



**Figure 7: Moore Machine State Diagram**

Table 1: State Table

Present	Next State		Output
Present	Next State		Output
State	x = 0	x = 1	Z
Initial	Initial	Got-1	0
Got-1	Initial	Got-11	0
Got-11	Initial	Got-111	0
Got-111	Initial	Got-111	1

Table 2: State Transition Table and Output Table

Present State	Next State		Output Z
	x = 0	x = 1	
Initial	Initial	Got-1	0
Got-1	Initial	Got-11	0
Got-11	Initial	Got-111	0
Got-111	Initial	Got-111	1

- We will use JK and D flip-flops for the Moore circuit implementation. The excitation tables for JK and D flip-flops (Table 3 & 4) are referenced to tabulate excitation table (See Table 5).

Table 3: Excitation Table for JK flip-flop

Q(t)	Q(t+1)	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Table 4: Excitation Table for D flip-flop

Q(t)	Q(t+1)	D
0	0	0
0	1	1
1	0	0
1	1	1

Table 5: Excitation Table for the Moore Implementation

Inputs of Comb.Circuits			Next State	Outputs of Comb.Circuit			Output	
Present State	Input			Flip-flop Inputs				
A	B	X	A	B	$J_A$	$K_A$	$D_B$	Z
0	0	0	0	0	0	X	0	0
0	0	1	0	1	0	X	1	0
0	1	0	0	0	0	X	0	0
0	1	1	1	0	1	X	0	0
1	0	0	0	0	X	1	0	0
1	0	1	1	1	X	0	1	0
1	1	0	0	0	X	1	0	1
1	1	1	1	1	X	0	1	1

- Simplifying Table 5 using maps, we get the following equations:

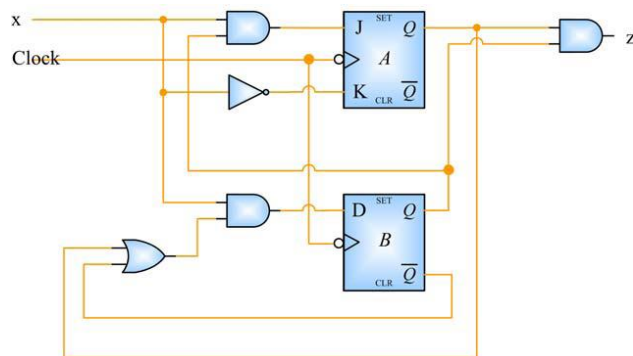
○  $J_A = X.B$

○  $K_A = X'$

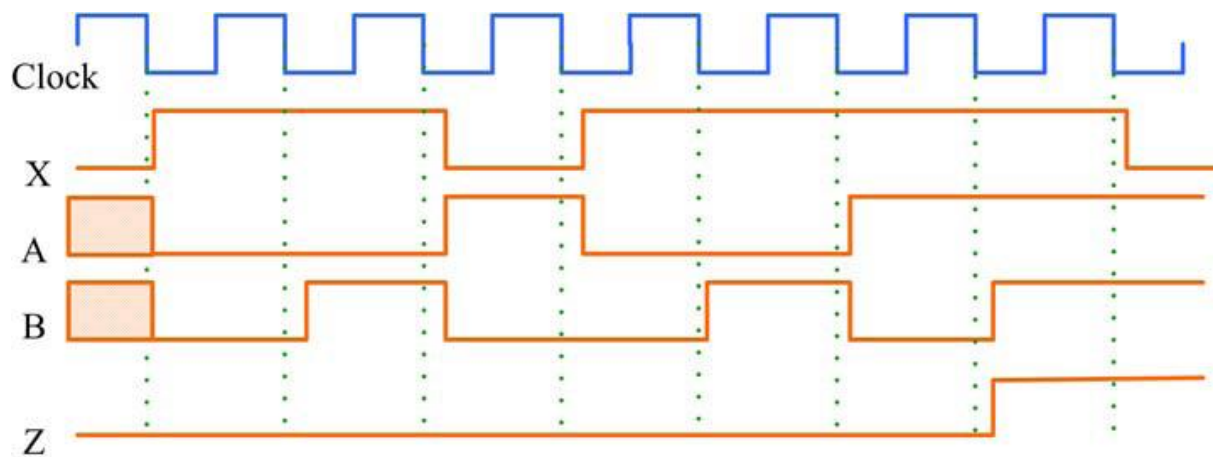
○  $D_B = X(A + B)$

○  $Z = A . B$

- Note that the output is a function of present state values only.
- The circuit diagram for Moore machine circuit implementation is shown in Figure 8.
- The timing diagram for Moore machine model is also shown in Figure 9.
- There is no false output in a Moore model, since the output depends only on the state of the flop flops, which are synchronized with clock. The outputs remain valid throughout the logic state in Moore model.



**Figure 8: Moore Machine Circuit Implementation for Sequence Detector.**



**Figure 9: Timing Diagram for Moore Model Sequence Detector.**



**Recommended question and answer –unit-7****Jan -2009**

Q.7 b) Give output function, excitation table and state transition diagram by analyzing the sequential circuit shown in Fig. 7.

(12)

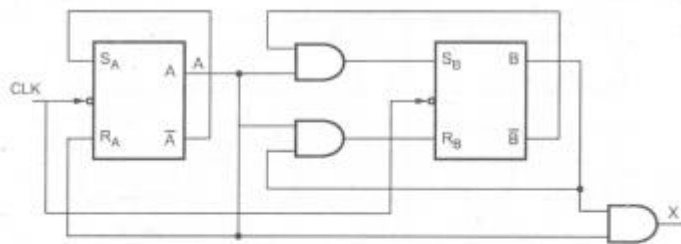


Fig. 7

Ans. : The excitation equations are :

$$S_A = \bar{A} \quad S_B = A\bar{B}$$

$$R_A = A \quad R_B = AB$$

The output equation is

$$X = AB$$

Evaluation of excitation and output expressions :

A	B	$S_A$	$R_A$	$S_B$	$R_B$	X
0	0	1	0	0	0	0

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

Excitation table :

Present state		Excitation $S_A, R_A, S_B, R_B$ for input	Output X
A	B		
0	0	1 0, 0 0	0
0	1	1 0, 0 0	0
1	0	0 1, 1 0	0
1	1	0 1, 0 1	1

Transition table :

Present state		Next state		Output X
A	B	$A^+$	$B^+$	
0	0	1	0	0
0	1	1	1	0
1	0	0	1	0
1	1	0	0	1

Let the states be  $S_0 = 00, S_1 = 01, S_2 = 10$  and  $S_3 = 11$

State table :

Present state	Next state	Output X
$S_0$	$S_2$	0
$S_1$	$S_3$	0
$S_2$	$S_1$	0
$S_3$	$S_0$	1

State diagram :

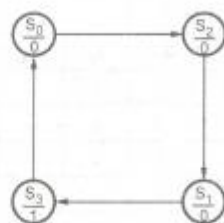


Fig. 7 (a)

**Jan -2008**

7.For the state machine M1 shown in Fig. 19 , obtain

- i) State table
- ii) Transition table
- iii) Excitation table for T flip-flop

iv) Logic circuit for T excitation realization

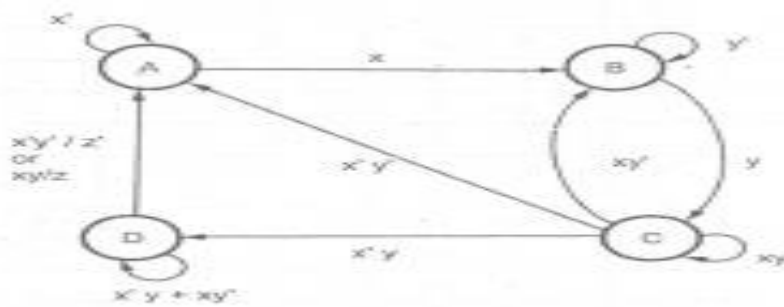


Fig- 19

Ans. : State table

Present state	Next state				Excitation table for T-FF		
	$\bar{x}\bar{y}$	$\bar{x}y$	$x\bar{y}$	$xy$	RS $Q_1$	NS $Q_2$	T
A	A, 0	A, 0	B, 0	B, 0	0	0	0
B	B, 0	C, 0	B, 0	C, 0	0	1	1
C	A, 0	D, 0	B, 0	C, 0	1	0	1
D	A, 0	D, 0	D, 0	A, 0	1	1	0
Let	A = 00	B = 01	C = 10	D = 11			

Transition table

Input	PS		NS		$T_2$	$T_1$
	P	Q	$P^{+1}$	$Q^{+1}$		
A	0	0	0	0	0	0
	0	1	0	0	0	0
	1	0	0	0	1	0
	1	1	0	0	1	0
B	0	0	0	1	0	0
	0	1	0	1	1	1
	1	0	0	1	0	0
C	1	1	0	1	1	1
	0	0	1	0	0	0
	0	1	1	0	1	1
	1	0	1	0	0	0
D	1	1	1	1	0	0
	0	0	1	1	1	1
	0	1	1	1	0	0
	1	0	1	1	0	0
	1	1	1	1	1	1

Using K-map we find the  $T_1$  and  $T_2$   
 For  $T_2$

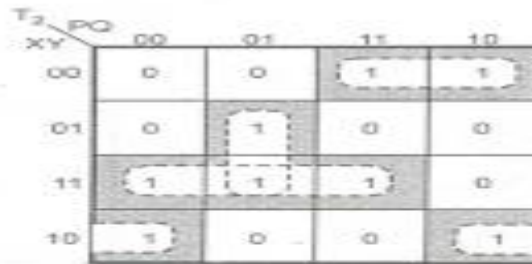


Fig. 20

$$T_2 = \bar{X}\bar{X}P + Y\bar{P}Q + XY\bar{P} + XYQ + XY\bar{Q}$$

$$= \bar{X}\bar{Y}P + Y\bar{P}Q + XY(\bar{P} + Q) + XY\bar{Q}$$

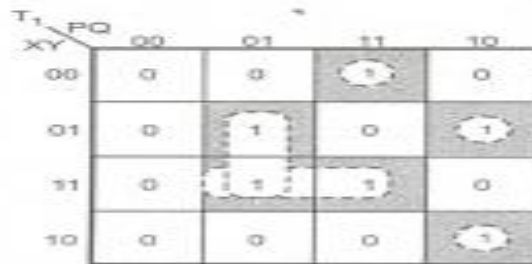
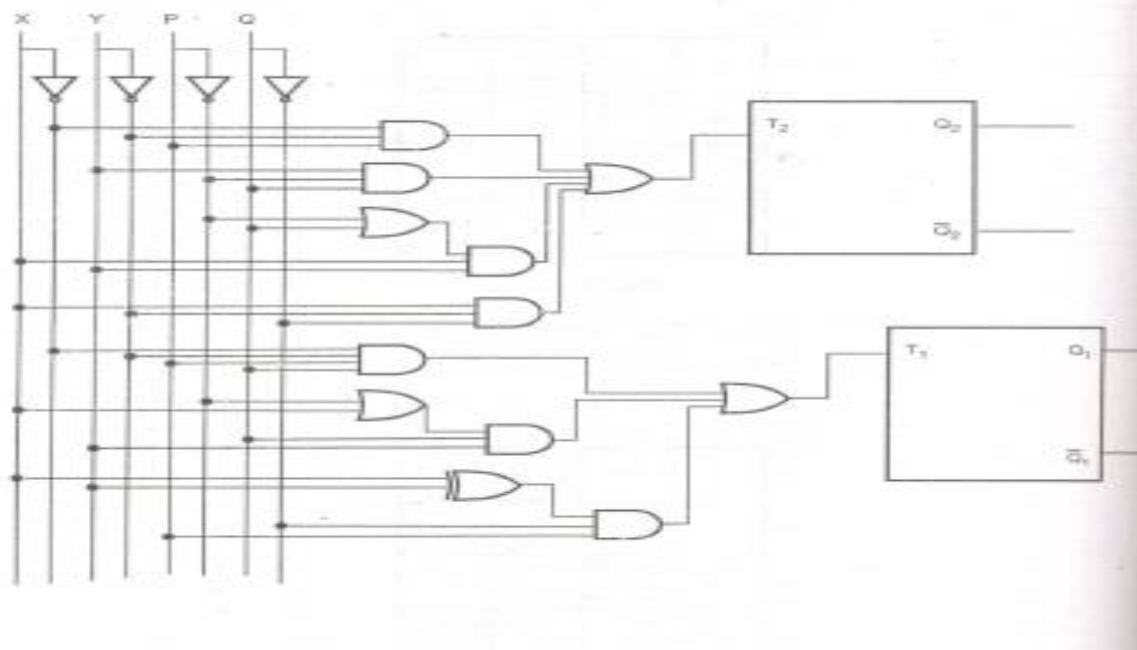


Fig. 21

$$T_1 = \bar{X}\bar{Y}PQ + Y\bar{P}Q + XYQ + X\bar{Y}P\bar{Q} + \bar{X}\bar{Y}P\bar{Q}$$

$$= \bar{X}\bar{Y}PQ + YQ(\bar{P} + X) + P\bar{Q}(X\bar{Y} + X\bar{Y})$$

$$= \bar{X}\bar{Y}PQ + YQ(\bar{P} + X) + P\bar{Q}(X \oplus Y)$$



c.State the rules for state assignments.

Ans. : Rules for state assignments

There are two basic rules for making state assignments.

Rule 1: States having the same NEXT STATES for a given input condition should have assignments which can be grouped into logically adjacent cells in a K-map.

Fig. 12 shows the example for Rule 1. As shown in the Fig. 12, there are four states whose next state is same. Thus states assignments for these states are 100, 101, 110 and 111, which can be grouped into logically adjacent cells in a K-map.

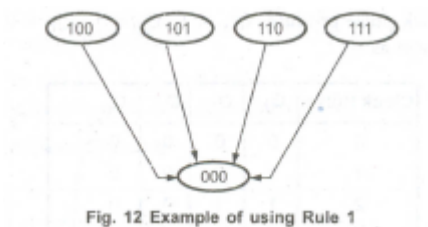


Fig. 12 Example of using Rule 1

Rule 2: States that are the NEXT STATES of a single state should have assignment which can be grouped into logically adjacent cells in a K-map.

Fig. 13 shows the example for Rule 2. As shown in the Fig. 13 for state 000, there are four next states. These states are assigned as 100, 101, 110 and 111 so that they can

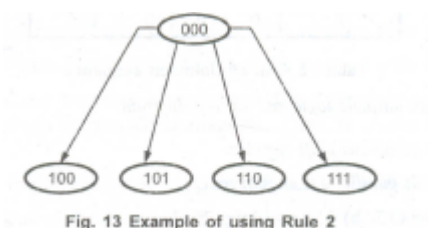


Fig. 13 Example of using Rule 2

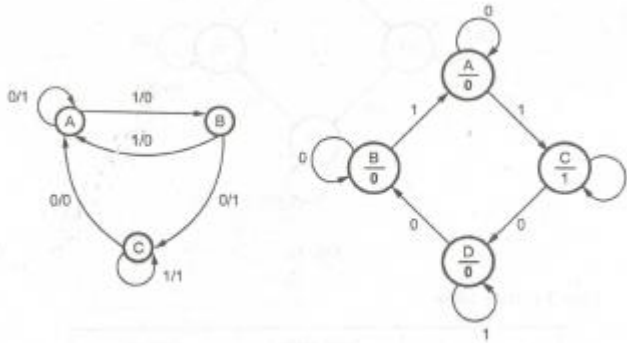
be grouped into logically adjacent cells in a K-map and table shows the state table with assigned states

Present State	Next state		Output	
	X = 0	X = 1	X = 0	X = 1
00	01	10	0	0
01	11	10	1	0
10	10	11	0	1
11	00	11	0	0

Table 4 State table with assigned states

**Aug 2008**

7. b) Construct the state table for the following state diagram.



Ans. :

Input x	Present state	Next state	Output Y
0	A	A	1
0	B	C	1
0	C	A	0

Input	Present state	Next state	Output
0	A	A	0
0	B	B	0
0	C	D	1

1	A	B	0
1	B	A	0
1	C	C	1

0	D	B	0
1	A	C	0
1	B	A	0
1	C	C	1
1	D	D	0

Aug-2007

applied to the J and K inputs of flip-flop C. Whenever both QA and QB are HIGH, the output of the AND gate makes the J and K inputs of flip-flop C HIGH, and flip-flop C toggles on the following clock pulse. At all other times, the J and K inputs of flip-flop C are held LOW by the AND gate output, and flip-flop does not change state.

b) Explain the different types of shift register. 5150, SIPO, PIPO, PISO with relevant circuit diagram. . [10]

Sol. :SISO Shift Register:

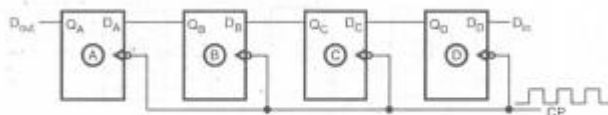


Fig. 13 Shift-left register

Fig. 13 shows serial in serial out shift-left register.

We will illustrate the entry of the four bit binary number 1111 into the register, beginning with the left-most bit.

Initially, register is cleared. SO

$$QAQBQCQO = 0000$$

a) When data 1 1 1 1 is applied serially, Le.

left-most 1 is applied as Din'

$$Din = 1, QAQBQCQo = 0000$$

The arrival of the first falling clock edge sets the right-most flip-flop, and the stored word becomes,

$$QAQBQCQO = 0001$$

b) When the next negative clock edge hits, the  $Q_A$  flip-flop sets and the register contents become,

$$QAQBQCQO = 0011$$

c) The third negative clock edge results in,

$$QAQBQCQO = 0111$$

d) The fourth falling clock edge gives,

$$QAQBQCQo = 1111$$

**SIPO Shift Register:** In this case, the data bits are entered into the register in the same manner as discussed in the last section, i.e. serially. But the output is taken in parallel. Once the data are stored, each bit appears on its respective output line and all bits are available simultaneously, instead of on a bit-by-bit basis as with the serial output as shown in Fig. 14.

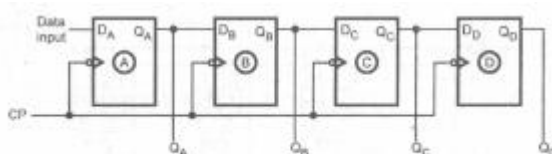


Fig. 14 A serial in parallel out shift register

**PIPO Shift Register :** From the third and second types of registers, it is cleared

that how to enter the data in parallel i.e. all bits simultaneously into the register and how to take data out in parallel from the register. In 'parallel in parallel out register', there is simultaneous entry of all data bits and the bits appear on parallel outputs simultaneously. Fig. 15 shows this type of register.

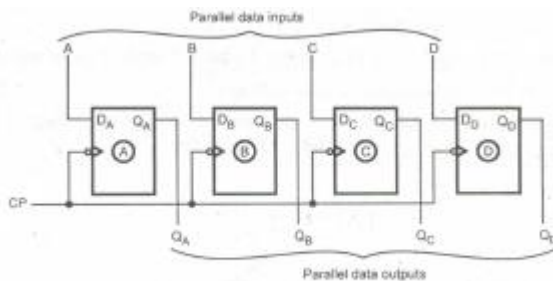


Fig. 15 Parallel in parallel out shift register

**PISO Shift Register :** In this type, the bits are entered in parallel i.e. simultaneously into their respective stages on parallel lines.

Fig. 16 illustrates a four-bit parallel in serial out register. There are four input lines  $X_A, X_B, X_C, X_D$  for entering data in parallel into the register. SHIFT/LOAD is the control input which allows shift or loading data operation of the register. When SHIFT/LOAD is low, gates  $G_1, G_2, G_3$  are enabled, allowing each input data bit to be applied to D input of its respective flip-flop. When a clock pulse is applied, the flip-flops with  $D = 1$  will SET and those with  $D = 0$  will RESET. Thus all four bits are stored simultaneously.

When SHIFT/LOAD is high, gates  $G_1, G_2, G_3$  are disabled and gates  $G_4, G_5, G_6$  are enabled. This allows the data bits to shift left from one stage to the next. The OR gates at the D-inputs of the flip-flops allow either the parallel data entry operation or shift operation, depending on which AND gates are enabled by the level on the SHIFT/LOAD input.



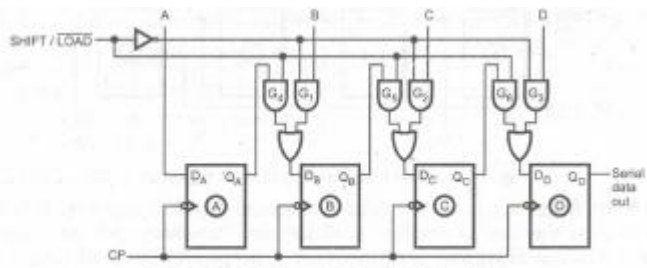


Fig. 16 Parallel in serial out shift register

notes4free.in

**Unit 8:**

6 Hours

**Sequential Design - II:**

Construction of state Diagrams, Counter Design

**Recommended readings:**

1. Donald D Givone, "Digital Principles and Design", Tata McGraw Hill Edition, 2002.

notes4free.in

Units- 6.4, 6.5

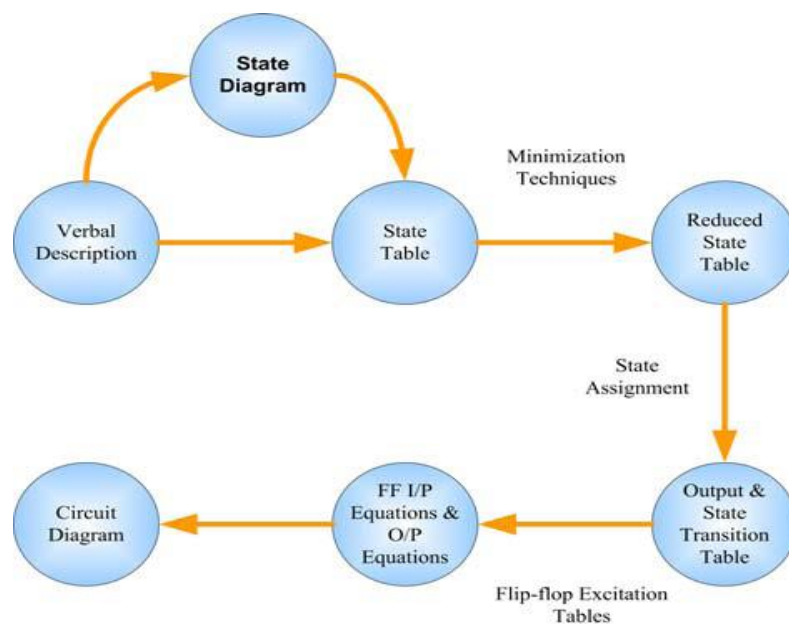
## Design of Synchronous Sequential Circuits

### Objectives

1. Design of synchronous sequential circuits with an example.
2. Construction of state diagrams and state tables/
3. Translation of State transition table into excitation table.
4. Logic diagram construction of a synchronous sequential circuit

### Sequential Circuit Design Steps

- □ The design of sequential circuit starts with verbal specifications of the problem (See Figure 1).



**Figure 1: Sequential Circuit Design Steps**

The next step is to derive the state table of the sequential circuit. A state table represents the verbal specifications in a tabular form.

In certain cases state table can be derived directly from verbal description of the problem.

In other cases, it is easier to first obtain a state diagram from the verbal description and then obtain the state table from the state diagram.

A state diagram is a graphical representation of the sequential circuit.

In the next step, we proceed by simplifying the state table by minimizing the number of states and obtain a reduced state table.

notes4free.in

The states in the reduced state table are then assigned binary-codes. The resulting table is called output and state transition table.

From the state transition table and using flip-flop's excitation tables, flip-flops input equations are derived. Furthermore, the output equations can readily be derived as well.

Finally, the logic diagram of the sequential circuit is constructed.

An example will be used to illustrate all these concepts.

### Sequence Recognizer

A sequence recognizer is to be designed to detect an input sequence of '1011'. The sequence recognizer outputs a '1' on the detection of this input sequence. The sequential circuit is to be designed using JK and D type flip-flops.

A sample input/output trace for the sequence detector is shown in Table 1.

**Table 1: Sample Input/Output Trace**

Input	0	1	1	0	1	0	1	1	0	1	1	1	0	1	0	1	1	1	0	0
Output	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	1	0	0	0

We will begin solving the problem by first forming a state diagram from the verbal description.

A state diagram consists of circles (which represent the states) and directed arcs that connect the circles and represent the transitions between states.

In a state diagram:

1. The number of circles is equal to the number of states. Every state is given a label (or a binary encoding) written inside the corresponding circle.
2. The number of arcs leaving any circle is  $2^n$ , where  $n$  is the number of inputs of the sequential circuit.
3. The label of each arc has the notation  $x/y$ , where  $x$  is the input vector that causes the state transition, and  $y$  is the value of the output during that present state.
4. An arc may leave a state and end up in the same or any other state.

Before we begin our design, the following should be noted

1. We do not have an idea about how many states the machine will have.
2. The states are used to “remember” something about the history of past inputs. For the sequence 1011, in order to be able to produce the output value 1 when the final 1 in the sequence is received, the circuit must be in a state that “remembers” that the previous three inputs were 101.
3. There can be more than one possible state machine with the same behavior.

### Deriving the State Diagram

Let us begin with an initial state (since a state machine must have at least one state) and denote it with ‘**S0**’ as shown in Figure 2 (a).

Two arcs leave state ‘**S0**’ depending on the input (being a 0 or a 1). If the input is a 0, then we return back to the same state. If the input is a 1, then we have to remember it (recall that we are trying to detect a sequence of 1011). We remember that the last input was a one by changing the state of the machine to a new state, say ‘**S1**’. This is illustrated in Figure 2 (b).

‘**S1**’ represents a state when the last single bit of the sequence was one. Outputs for both transitions are zero, since we have not detected what we are looking for.

Again in state ‘**S1**’, we have two outgoing arcs. If the input is a 1, then we return to the same state and if the input is a 0, then we have to remember it (second number in the sequence). We can do so by transiting to a new state, say ‘**S2**’. This is illustrated in Figure 2 (c).

Note that if the input applied is '1', the next state is still '**S1**' and not the initial state '**S0**'. This is because we take this input 1 as the first digit of new sequence. The output still remains 0 as we have not detected the sequence yet.

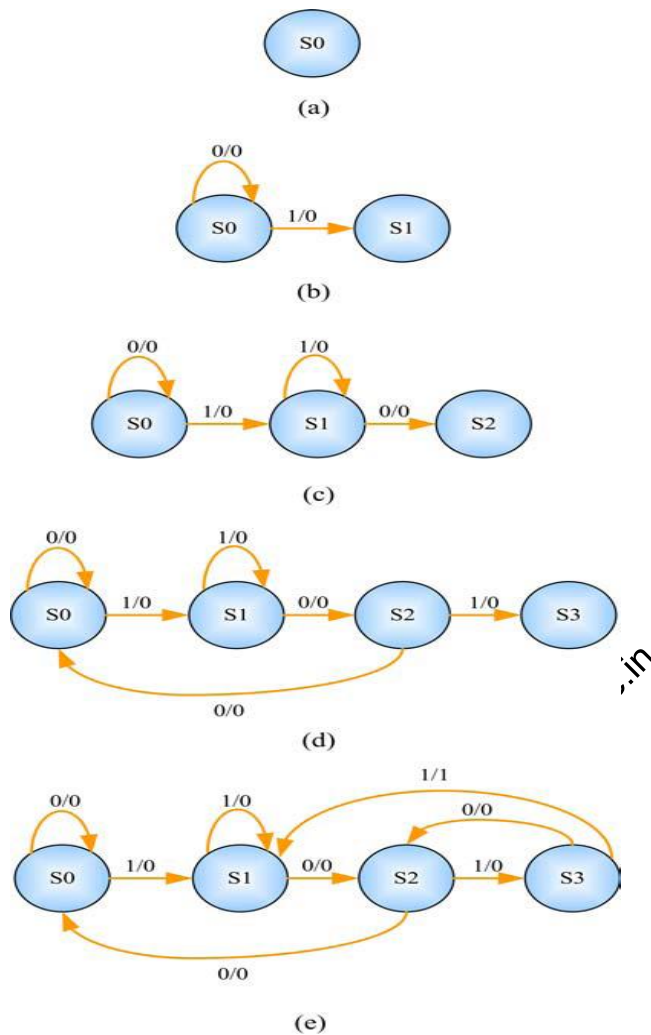
State '**S2**' represents detection of '10' as the last two bits of the sequence. If now the input is a '1', we have detected the third bit in our sequence and need to remember it. We remember it by transiting to a new state, say '**S3**' as shown in Figure 2 (d). If the input is '0' in state '**S2**' then it breaks the sequence and we need to start all over again. This is achieved by transiting to initial state '**S0**'. The outputs are still 0.

In state '**S3**', we have detected input sequence '101'. Another input 1 completes our detection sequence as shown in Figure 2 (e). This is signaled by an output 1. However we transit to state '**S1**' instead of '**S0**' since this input 1 can be counted as first 1 of a new sequence. Application of input 0 to state '**S3**' means an input sequence of 1010. This implies the last two bits in the sequence were 10 and we transit to a state that remembers this input sequence, i.e. state '**S2**'. Output remains as zero.

notes4free.in



Figure 2: Deriving the State Diagram of the Sequence Recognizer



**Deriving the State Table**

A state table represents time sequence of inputs, outputs, and states in a tabular form. The state table for the previous state diagram is shown in Table 2.

The state table can also be represented in an alternate form as shown in Table 3.

Here the present state and inputs are tabulated as inputs to the combinational circuit. For every combination of present state and input, next state column is filled from the state table.

The number of flip-flops required is equal to  $\lceil \log_2(\text{number of states}) \rceil$ .

Thus, the state machine given in the figure will require two flip-flops  $\lceil \log_2(4) \rceil = 2$ . We assign letters A and B to them.

Table 2: State Table of the Sequence Recognizer

Present State	Next State		Output	
	X=0	X=1	X=0	X=1
S0	S0	S1	0	0
S1	S2	S1	0	0
S2	S0	S3	0	0
S3	S2	S1	0	1

Table 3: Alternative Format of Table 2

Inputs of Combinational Circuit		Next State	Output
Present State	Input		
S0	0	S0	0
S0	1	S1	0
S1	0	S2	0
S1	1	S1	0
S2	0	S0	0
S2	1	S3	0
S3	0	S2	0
S3	1	S1	1

### State Assignment

The states in the constructed state diagram have been assigned symbolic names rather than binary codes.

It is necessary to replace these symbolic names with binary codes in order to proceed with the design.

In general, if there are  $m$  states, then the codes must contain  $n$  bits, where  $2^n \geq m$ , and each state must be assigned a unique code.

There can be many possible assignments for our state machine. One possible assignment is shown in Table 4.

**Table 4: State Assignment**

State	Assignment
S0	00
S1	01
S2	10
S3	11

- The assignment of state codes to states results in state transition table as shown.
- It is important to mention here that the binary code of the present state at a given time  $t$  represents the values stored in the flip-flops; and the next-state represents the values of the flip-flops one clock period later, at time  $t+1$ .

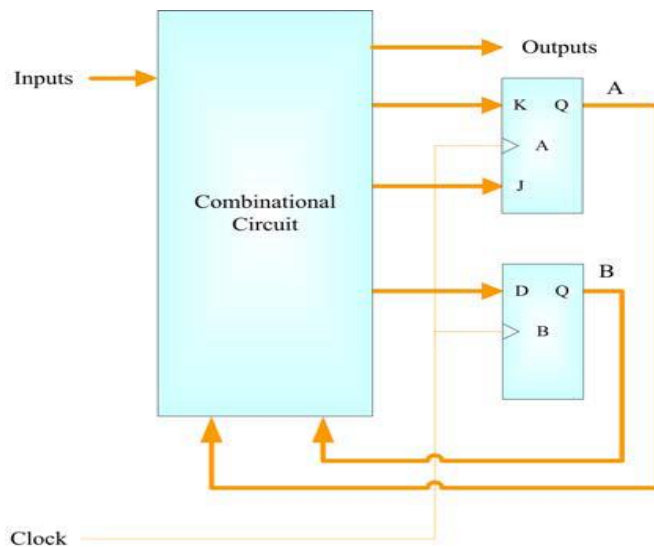
**Table 5: State Transition Table**

Inputs of Combinational Circuit		Next State	Output
Present State	Input		
A	B	X	Y
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

### General Structure of Sequence Recognizer

- The specifications required using JK and D type flip-flops.

Referring to the general structure of sequential circuit shown in Figure 3, our synthesized circuit will look like that as shown in the figure. Observe the feedback paths.



**Figure 3: General Structure of the Sequenc Recognizer**

- What remains to be determined is the combinational circuit which specifies the external outputs and the flip-flop inputs.
- The state transition table as shown can now be expanded to construct the excitation table for the circuit.
- Since we are designing the sequential circuit using JK and D type flip-flops, we need to correlate the required transitions in state transition table with the excitation tables of JK and D type-flip-flops.
- The functionality of the required combinational logic is encapsulated in the excitation table. Thus, the excitation table is next simplified using map or other simplification methods to yield Boolean expressions for inputs of the used flip-flops as well as the circuit outputs.

### Deriving the Excitation Table

- The excitation table (See Table 6) describes the behavior of the combinational portion of sequential circuit.

**Table 6: Excitation Table of the Sequence Recognizer**

Present State		Input	Flip-flops Inputs					
A	B	X	A	B	Y	$J_A$	$K_A$	$D_B$
0	0	0	0	0	0	0	X	0
0	0	1	0	1	0	0	X	1
0	1	0	1	0	0	1	X	0
0	1	1	0	1	0	0	X	1
1	0	0	0	0	0	X	1	0
1	0	1	1	1	0	X	0	1
1	1	0	1	0	0	X	0	0
1	1	1	0	1	1	X	1	1

- For deriving the actual circuitry for the combinational circuit, we need to simplify the excitation table in a similar way we used to simplify truth tables for purely combinational circuits.
- Whereas in combinational circuits, our concern were only circuit outputs; in sequential circuits, the combinational circuitry also feeding the flip-flops inputs. Thus, we need to simplify the excitation table for both outputs as well as flip-flops inputs.
- We can simplify flip-flop inputs and output using K-maps as shown in Figure 4.
- Finally the logic diagram of the sequential circuit can be made as shown in Figure 5.

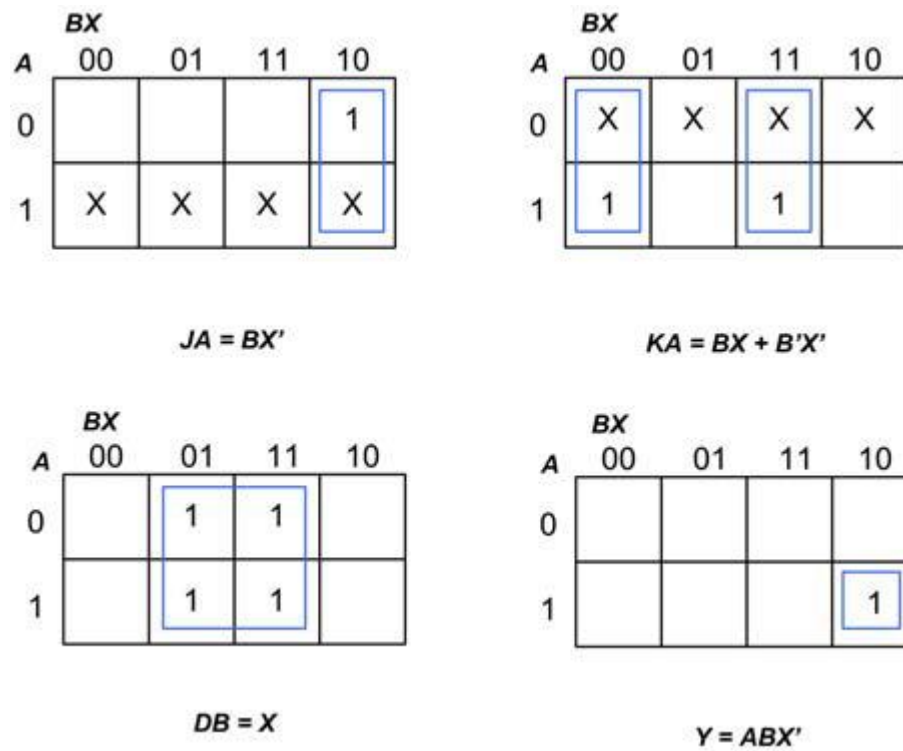


Figure 4: Input Equations of the Sequence Recognizer

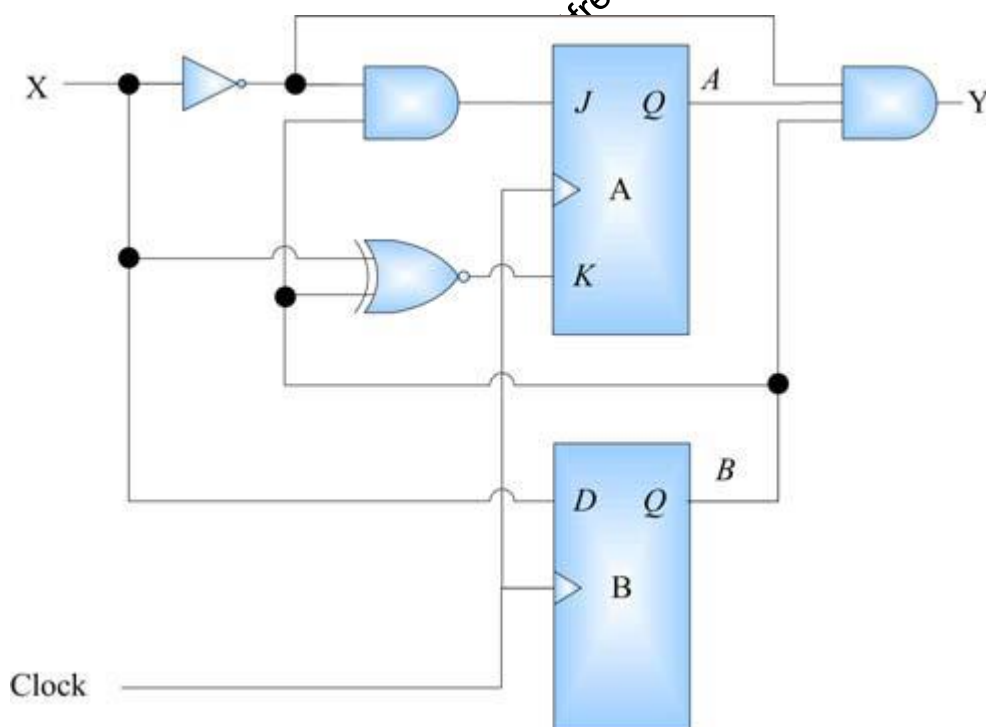


Figure 5: Circuit Diagram of the Sequence Recognizer

**Recommended question and answer –unit-8**

Q.8 a) Design a cyclic mod-6 synchronous binary counter ? state diagram, transition table using JK flip-flop. (10)

Ans. : Design of a synchronous mod-6 counter using clocked JK flip-flops The counter with n flip-flops has maximum mod number  $2^n$ . For example, 3-bit binary counter is a mod 8 counter. This basic counter can be modified to produce MOD numbers less than  $2^n$  by allowing the counter to skip states those are normally

part of counting sequence. Let us design mod-6 counter using clocked JK flip-flops.

Step 1 : Find number of flip-flops required to build the counter : Flip-Flops required are:  $2^n \geq N$ .

Here  $N = 6 \therefore n = 3$

i.e. three flip-flops are required.

Step 2 : Write an excitation table for JK flip-flop.

$Q_n$	$Q_{n+1}$	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Table 2

Step 3 : Determine the transition table.

Present state			Next state			Flip-flop inputs					
$Q_A$	$Q_B$	$Q_C$	$Q_{A+1}$	$Q_{B+1}$	$Q_{C+1}$	$J_A$	$K_A$	$J_B$	$K_B$	$J_C$	$K_C$
0	0	0	0	0	1	0	x	0	x	1	x
0	0	1	0	1	0	0	x	1	x	x	1
0	1	0	0	1	1	0	x	x	0	1	x
0	1	1	1	0	0	1	x	x	1	x	1
1	0	0	1	0	1	x	0	0	x	1	x
1	0	1	0	0	0	x	1	0	x	x	1
1	1	0	x	x	x	x	x	x	x	x	x
1	1	1	x	x	x	x	x	x	x	x	x

Step 4 : K-map simplification for flip-flop inputs.

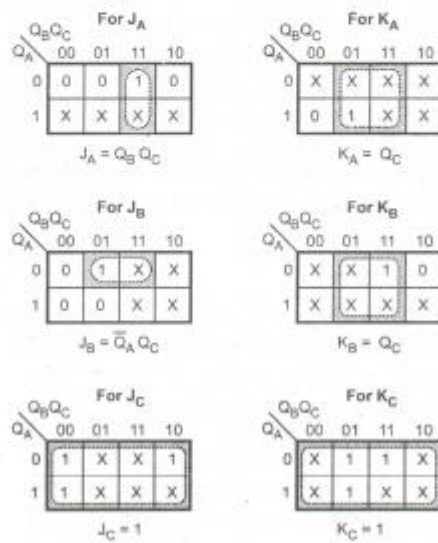


Fig. 8



Step 5 : Implement the counter.

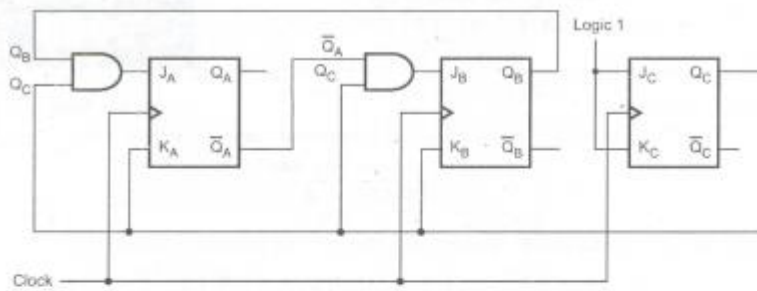


Fig. 9 Implementation of mod-6 synchronous counter

Note : To avoid crossing of lines and to have better clarity the circuit can also be represented as shown in Fig. 10. Here, instead of actual connections only signal names are specified.

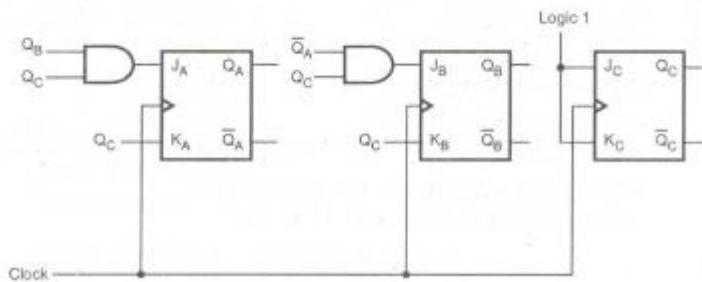


Fig. 10 Simplified representation of Fig. 9

notes4fre

**Jan-2008**

**Q.8 a)** Construct a mealy state diagram that will detect a serial sequence of 10110. When the input pattern has been detected, cause an output Z to be asserted high. I

Ans. :

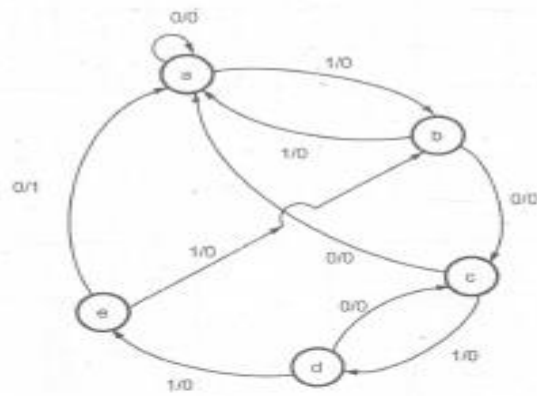


Fig. 23

State table

Present state	Next state		Output	
	x = 0	x = 1	x = 0	x = 1
a	a	b	0	0
b	c	a	0	0
c	a	d	0	0
d	c	e	0	0
e	a	b	1	0

Excitation table

Present state			Next state						Output	
			A*	B*	C*	A*	B*	C*		
A	B	C	x = 0			x = 1			x = 0	x = 1
0	0	0	0	0	0	0	0	1	0	0
0	0	1	0	1	0	0	0	0	0	0

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

K-map simplification

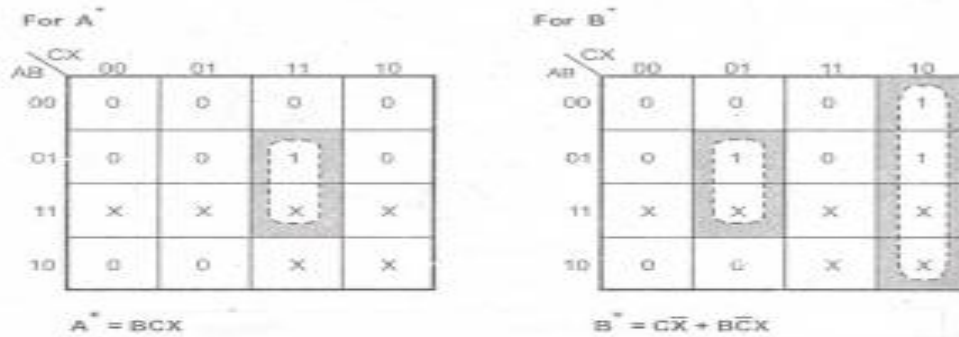
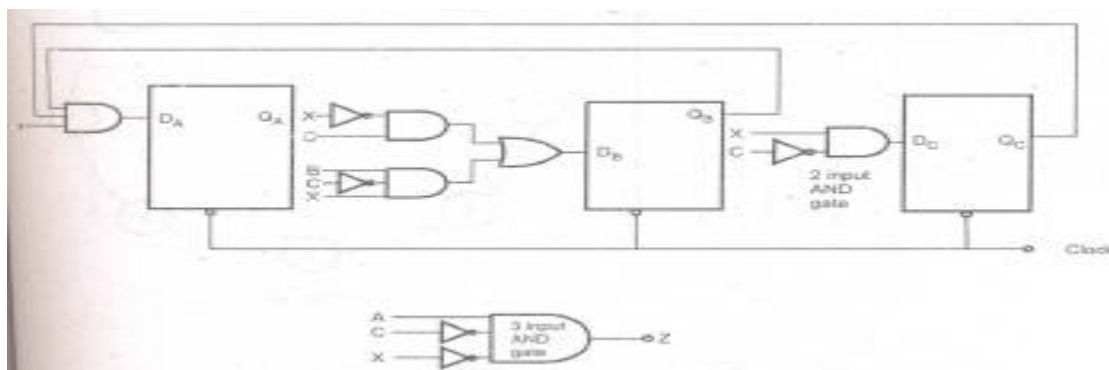
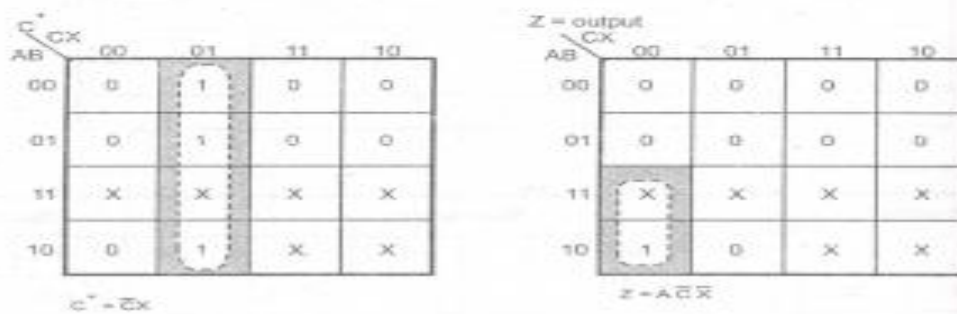


Fig. 24



b) Design a cyclic modulo-8 synchronous counter using J-K flip-flop that will count the number of occurrences of an input; that is, the number of times it is a 1. The input variable X must be coincident with the clock to be counted. The counter is to count in binary.

(12)

Present state			Next state			$J_A$	$K_A$	$J_B$	$K_B$	$J_C$	$K_C$
A	B	C	$A_{+1}$	$B_{+1}$	$C_{+1}$						
0	0	0	0	0	1	0	X	0	X	1	X
0	0	1	0	1	0	0	X	1	X	X	1
0	1	0	0	1	1	0	X	X	0	1	X
0	1	1	1	0	0	1	X	X	1	X	1
1	0	0	1	0	1	X	0	0	X	1	X
1	0	1	1	1	0	X	0	1	X	X	1
1	1	0	1	1	1	X	0	X	0	1	X
1	1	1	0	0	0	X	1	X	1	X	1

Excitation table

Present	Next	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

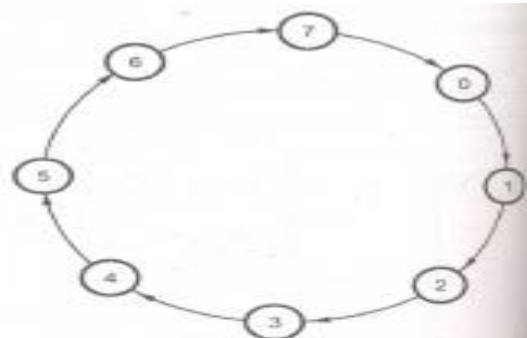


Fig. 27

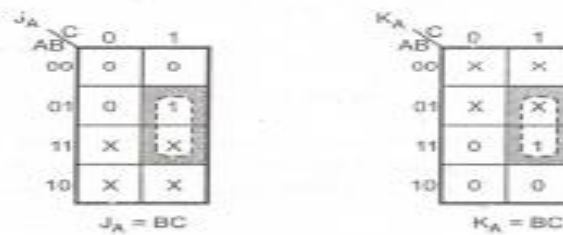


Fig. 28

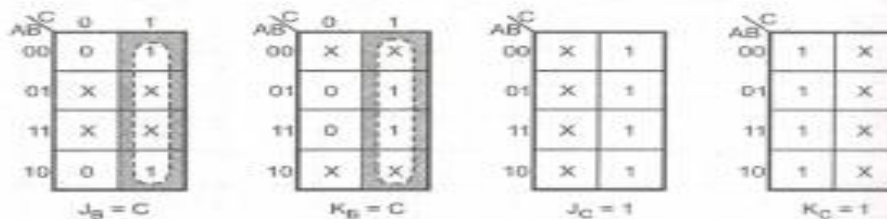


Fig. 29

**Aug 2009**

c.State the rules for state assignments.

Ans. : Rules for state assignments

There are two basic rules for making state assignments.

Rule 1: States having the same NEXT STATES for a given input condition should have assignments which can be grouped into logically adjacent cells in a K-map.

Fig. 12 shows the example for Rule 1. As shown in the Fig. 12, there are four states whose next state is same. Thus states assignments for these states are 100,

101, 110 and 111, which can be grouped into logically adjacent cells in a K-map.

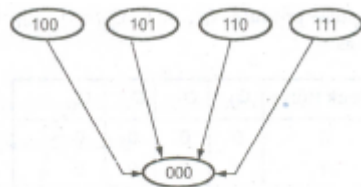


Fig. 12 Example of using Rule 1

Rule 2: States that are the NEXT STATES of a single state should have assignment which can be grouped into logically adjacent cells in a K-map.

Fig. 13 shows the example for Rule 2. As shown in the Fig. 13 for state 000, there

are four next states. These states are assigned as 100, 101, 110 and 111 so that they can

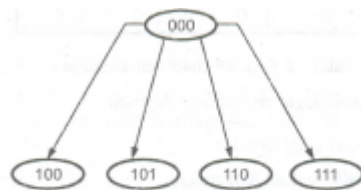


Fig. 13 Example of using Rule 2

be grouped into logically adjacent cells in a K-map and table shows the state table with assigned states

Present State	Next state		Output	
	X = 0	X = 1	X = 0	X = 1
00	01	10	0	0
01	11	10	1	0
10	10	11	0	1
11	00	11	0	0

Table 4 State table with assigned states

**Aug-2008**

Q.8 a) Design a clocked sequential circuit that operates according to the state diagram

shown. Implement the circuit using D flip-flop. (12)

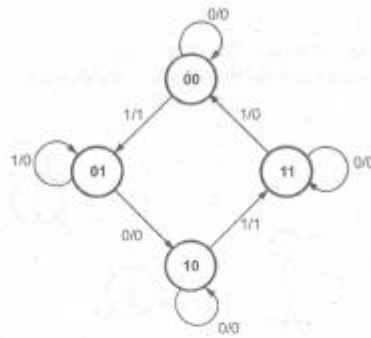


Fig. 12

Ans. : Step 1 : State table

Present state		Next state		Output	
		x = 0	x = 1	x = 0	x = 1
A	B	A* B*	A* B*	Y	Y
0	0	X X	0 1	X	1
0	1	1 0	0 1	0	0
1	0	1 0	1 1	0	1
1	1	1 1	0 0	0	0

Step 2 : Design using D flip-flop

Q <sub>n</sub>	Q <sub>n+1</sub>	D
0	0	0
0	1	1
1	0	0
1	1	1

Present state		Next state				Output			
		x = 0		x = 1		x = 0		x = 1	
A	B	A*	B*	A*	B*	D <sub>A</sub>	D <sub>B</sub>	D <sub>A</sub>	D <sub>B</sub>
0	0	X	X	0	1	X	X	0	1
0	1	1	0	0	1	1	0	0	1
1	0	1	0	1	1	1	0	1	1
1	1	1	1	0	0	1	1	0	0

K-map simplification :

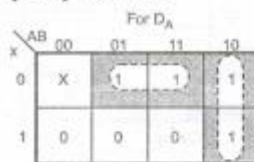


Fig. 12 (a)

$$D_A = \bar{x} B + A\bar{B}$$

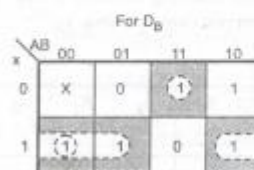


Fig. 12 (b)

$$D_B = \bar{x} AB + x\bar{A} + x\bar{B}$$

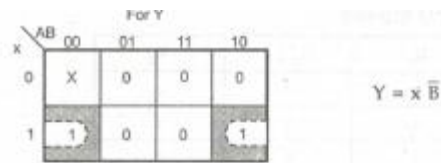


Fig. 12 (c)

Sequential circuit :

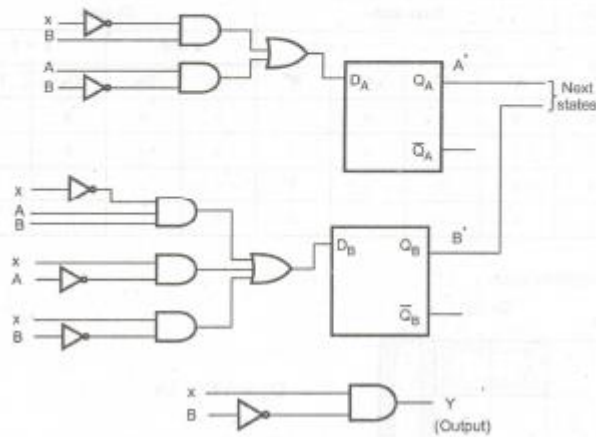


Fig. 12 (d)

b) Design a counter using JK flip-flops whose counting sequence is 000, 001, 100, 110, 111, 101, 000 etc. by obtaining its minimal sum equations. (8)

Ans. :

Present state			Next State			J <sub>1</sub>	K <sub>1</sub>	J <sub>2</sub>	K <sub>2</sub>	J <sub>3</sub>	K <sub>3</sub>
C <sub>n</sub>	B <sub>n</sub>	A <sub>n</sub>	C <sub>n+1</sub>	B <sub>n+1</sub>	A <sub>n+1</sub>						
0	0	0	0	0	1	0	X	0	X	1	X
0	0	1	1	0	0	1	X	0	X	X	1

0	1	0	X	X	X	X	X	X	X	X	X
0	1	1	X	X	X	X	X	X	X	X	X
1	0	0	1	1	0	X	0	1	X	0	X
1	0	1	0	0	0	X	1	0	X	X	1
1	1	0	1	1	1	X	0	X	0	1	X
1	1	1	1	0	1	X	0	X	1	X	0

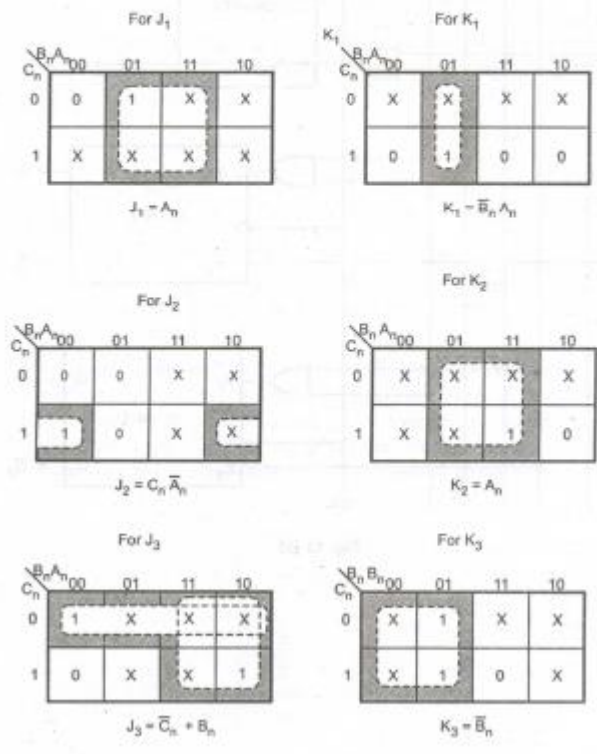


Fig. 13 (a)

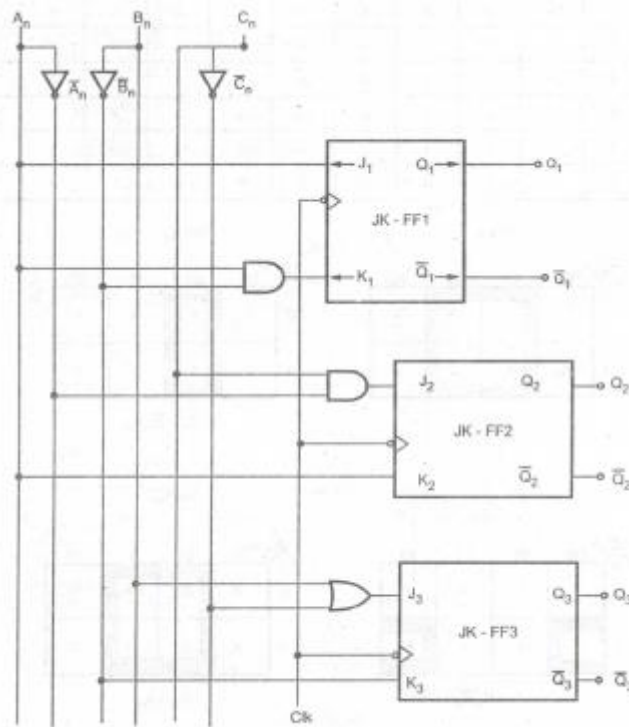


Fig. 13 (b)



Aug-2007

Q.8 a) A combinational circuit is defined by the functions :

$$F_1 = \Sigma m(3, 5, 7), \quad F_2 = \Sigma m(4, 5, 7)$$

Implement the circuit with a PLA having 3 inputs, 3 product terms and 2 outputs.[8]

Sol. : This topic is not included in the new syllabus.

b) A sequential network has one input and one output. The state diagram is shown in Fig. 17. Design the sequential circuit with T flip flop. [12]

Sol. :

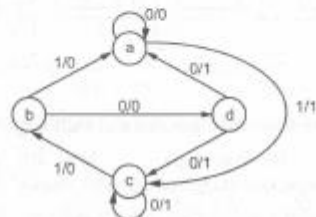


Fig. 17

Input	Present state		Next state		Flip-Flop inputs		Output
	A	B	A <sup>*</sup>	B <sup>*</sup>	T <sub>A</sub>	T <sub>B</sub>	
0	0	0	0	0	0	0	0
0	0	1	1	1	1	0	0
0	1	0	1	0	0	0	1
0	1	1	0	0	1	1	1
1	0	0	1	0	1	0	1
1	0	1	0	0	0	1	0
1	1	0	0	1	1	1	0
1	1	1	1	0	0	1	0

Logic diagram

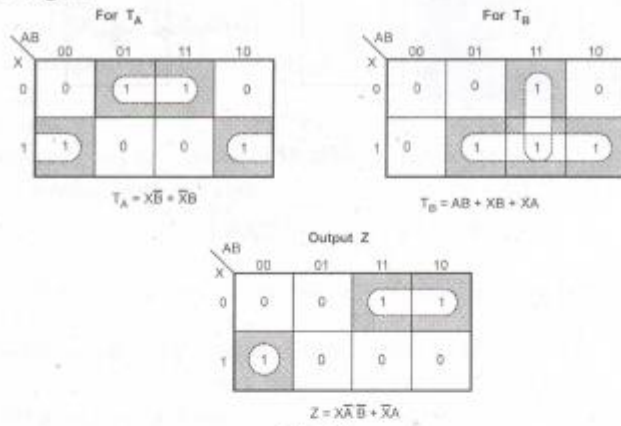


Fig. 18

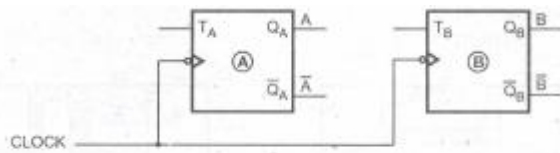
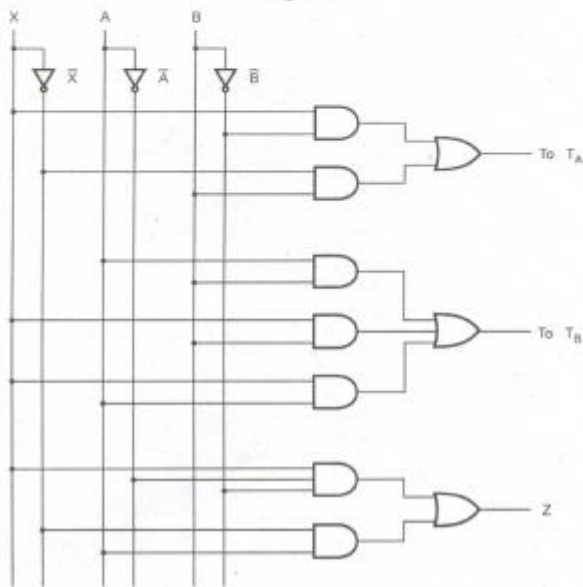


Fig. 19