

# NUMBER SYSTEM AND CODES

## INTRODUCTION:-

- The term digital refers to a process that is achieved by using discrete unit.
- In number system there are different symbols and each symbol has an absolute value and also has place value.

## RADIX OR BASE:-

The radix or base of a number system is defined as the number of different digits which can occur in each position in the number system.

## RADIX POINT :-

The generalized form of a decimal point is known as radix point. In any positional number system the radix point divides the integer and fractional part.

$$N_r = [ \text{Integer part} \cdot \text{Fractional part} ]$$

↑  
Radix point

## NUMBER SYSTEM:-

In general a number in a system having base or radix ' r ' can be written as

$$a_n \ a_{n-1} \ a_{n-2} \ \dots \ a_0 \ . \ a_{-1} \ a_{-2} \ \dots \ a_{-m}$$

This will be interpreted as

$$Y = a_n \times r^n + a_{n-1} \times r^{n-1} + a_{n-2} \times r^{n-2} + \dots + a_0 \times r^0 + a_{-1} \times r^{-1} + a_{-2} \times r^{-2} + \dots + a_{-m} \times r^{-m}$$

where Y = value of the entire number

$a_n$  = the value of the  $n^{\text{th}}$  digit

r = radix

## TYPES OF NUMBER SYSTEM:-

There are four types of number systems. They are

1. Decimal number system
2. Binary number system
3. Octal number system
4. Hexadecimal number system

## DECIMAL NUMBER SYSTEM:-

- The decimal number system contain ten unique symbols 0,1,2,3,4,5,6,7,8 and 9.
- In decimal system 10 symbols are involved, so the base or radix is 10.
- It is a positional weighted system.
- The value attached to the symbol depends on its location with respect to the decimal point.

In general,

$$d_n \ d_{n-1} \ d_{n-2} \ \dots \ d_0 \ . \ d_{-1} \ d_{-2} \ \dots \ d_{-m}$$

is given by

$$(d_n \times 10^n) + (d_{n-1} \times 10^{n-1}) + (d_{n-2} \times 10^{n-2}) + \dots + (d_0 \times 10^0) + (d_{-1} \times 10^{-1}) + (d_{-2} \times 10^{-2}) + \dots + (d_{-m} \times 10^{-m})$$

**For example:-**

$$\begin{aligned} 9256.26 &= 9 \times 1000 + 2 \times 100 + 5 \times 10 + 6 \times 1 + 2 \times (1/10) + 6 \times (1/100) \\ &= 9 \times 10^3 + 2 \times 10^2 + 5 \times 10^1 + 6 \times 10^0 + 2 \times 10^{-1} + 6 \times 10^{-2} \end{aligned}$$

### **BINARY NUMBER SYSTEM:-**

- The binary number system is a positional weighted system.
- The base or radix of this number system is 2.
- It has two independent symbols.
- The symbols used are 0 and 1.
- A binary digit is called a bit.
- The binary point separates the integer and fraction parts.

In general,

$$d_n \ d_{n-1} \ d_{n-2} \ \dots \ d_0 \ . \ d_{-1} \ d_{-2} \ \dots \ d_{-k}$$

is given by

$$(d_n \times 2^n) + (d_{n-1} \times 2^{n-1}) + (d_{n-2} \times 2^{n-2}) + \dots + (d_0 \times 2^0) + (d_{-1} \times 2^{-1}) + (d_{-2} \times 2^{-2}) + \dots + (d_{-k} \times 2^{-k})$$

### **OCTAL NUMBER SYSTEM:-**

- It is also a positional weighted system.
- Its base or radix is 8.
- It has 8 independent symbols 0,1,2,3,4,5,6 and 7.
- Its base  $8 = 2^3$ , every 3-bit group of binary can be represented by an octal digit.

### **HEXADECIMAL NUMBER SYSTEM:-**

- The hexadecimal number system is a positional weighted system.
- The base or radix of this number system is 16.
- The symbols used are 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E and F
- The base  $16 = 2^4$ , every 4-bit group of binary can be represented by an hexadecimal digit.

### **CONVERSION FROM ONE NUMBER SYSTEM TO ANOTHER :-**

#### **1. BINARY NUMBER SYSTEM:-**

##### **(a) Binary to decimal conversion:-**

In this method, each binary digit of the number is multiplied by its positional weight and the product terms are added to obtain decimal number.

**For example:**

**(i) Convert  $(10101)_2$  to decimal.**

**Solution :**

$$\begin{aligned} \text{(Positional weight)} & \quad 2^4 \ 2^3 \ 2^2 \ 2^1 \ 2^0 \\ \text{Binary number} & \quad 10101 \\ & = (1 \times 2^4) + (0 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) \\ & = 16 + 0 + 4 + 0 + 1 \\ & = (21)_{10} \end{aligned}$$

**(ii) Convert  $(111.101)_2$  to decimal.**

**Solution:**

$$\begin{aligned} (111.101)_2 & = (1 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) + (1 \times 2^{-1}) + (0 \times 2^{-2}) + (1 \times 2^{-3}) \\ & = 4 + 2 + 1 + 0.5 + 0 + 0.125 \\ & = (7.625)_{10} \end{aligned}$$

**(b) Binary to Octal conversion:-**

For conversion binary to octal the binary numbers are divided into groups of 3 bits each, starting at the binary point and proceeding towards left and right.

| <u>Octal</u> | <u>Binary</u> | <u>Octal</u> | <u>Binary</u> |
|--------------|---------------|--------------|---------------|
| 0            | 000           | 4            | 100           |
| 1            | 001           | 5            | 101           |
| 2            | 010           | 6            | 110           |
| 3            | 011           | 7            | 111           |

**For example:**

**(i) Convert  $(101111010110.110110011)_2$  into octal.**

**Solution :**

Group of 3 bits are                      101   111   010   110   .   110   110   011

Convert each group into octal =    5     7     2     6     .     6     6     3

The result is  **$(5726.663)_8$**

**(ii) Convert  $(10101111001.0111)_2$  into octal.**

**Solution :**

Binary number                      10   101   111   001   .   011   1

Group of 3 bits are                = 010   101   111   001   .   011   100

Convert each group into octal =    2     5     7     1     .     3     4

The result is  **$(2571.34)_8$**

**(c) Binary to Hexadecimal conversion:-**

For conversion binary to hexadecimal number the binary numbers starting from the binary point, groups are made of 4 bits each, on either side of the binary point.

| <u>Hexadecimal</u> | <u>Binary</u> | <u>Hexadecimal</u> | <u>Binary</u> |
|--------------------|---------------|--------------------|---------------|
| 0                  | 0000          | 8                  | 1000          |
| 1                  | 0001          | 9                  | 1001          |
| 2                  | 0010          | A                  | 1010          |
| 3                  | 0011          | B                  | 1011          |
| 4                  | 0100          | C                  | 1100          |
| 5                  | 0101          | D                  | 1101          |
| 6                  | 0110          | E                  | 1110          |
| 7                  | 0111          | F                  | 1111          |

**For example:**

**(i) Convert  $(1011011011)_2$  into hexadecimal.**

**Solution:**

Given Binary number                    10    1101   1011

Group of 4 bits are                    0010   1101   1011

Convert each group into hex = 2        D        B

The result is  $(2DB)_{16}$

**(ii) Convert  $(01011111011.011111)_2$  into hexadecimal.**

**Solution:**

Given Binary number                    010   1111   1011   .   0111   11

Group of 3 bits are                    = 0010   1111   1011   .   0111   1100

Convert each group into octal = 2        F        B        .        7        C

The result is  $(2FB.7C)_{16}$

## **2. DECIMAL NUMBER SYSTEM:-**

### **(a)Decimal to binary conversion:-**

In the conversion the integer number are converted to the desired base using successive division by the base or radix.

**For example:**

**(i) Convert  $(52)_{10}$  into binary.**

**Solution:**

Divide the given decimal number successively by 2 read the integer part remainder upwards to get equivalent binary number. Multiply the fraction part by 2. Keep the integer in the product as it is and multiply the new fraction in the product by 2. The process is continued and the integer are read in the products from top to bottom.

$$\begin{array}{r}
 2 \overline{)52} \\
 2 \overline{)26} \quad - 0 \\
 2 \overline{)13} \quad - 0 \\
 2 \overline{)6} \quad - 1 \\
 2 \overline{)3} \quad - 0 \\
 2 \overline{)1} \quad - 1 \\
 0 \quad - 1
 \end{array}$$

Result of  $(52)_{10}$  is  $(110100)_2$

(ii) Convert  $(105.15)_{10}$  into binary.

Solution:

| Integer part                  | Fraction part          |
|-------------------------------|------------------------|
| $2 \overline{) 105}$          | $0.15 \times 2 = 0.30$ |
| $2 \overline{) 52} \quad - 1$ | $0.30 \times 2 = 0.60$ |
| $2 \overline{) 26} \quad - 0$ | $0.60 \times 2 = 1.20$ |
| $2 \overline{) 13} \quad - 0$ | $0.20 \times 2 = 0.40$ |
| $2 \overline{) 6} \quad - 1$  | $0.40 \times 2 = 0.80$ |
| $2 \overline{) 3} \quad - 0$  | $0.80 \times 2 = 1.60$ |
| $2 \overline{) 1} \quad - 1$  |                        |
| $0 \quad - 1$                 |                        |

Result of  $(105.15)_{10}$  is  $(1101001.001001)_2$

### **(b) Decimal to octal conversion:-**

To convert the given decimal integer number to octal, successively divide the given number by 8 till the quotient is 0. To convert the given decimal fractions to octal successively multiply the decimal fraction and the subsequent decimal fractions by 8 till the product is 0 or till the required accuracy is obtained.

For example:

(i) Convert  $(378.93)_{10}$  into octal.

Solution:

|                               |                        |
|-------------------------------|------------------------|
| $8 \overline{) 378}$          | $0.93 \times 8 = 7.44$ |
| $8 \overline{) 47} \quad - 2$ | $0.44 \times 8 = 3.52$ |
| $8 \overline{) 5} \quad - 7$  | $0.52 \times 8 = 4.16$ |
| $0 \quad - 5$                 | $0.16 \times 8 = 1.28$ |

Result of  $(378.93)_{10}$  is  $(572.7341)_8$

### **(c) Decimal to hexadecimal conversion:-**

The decimal to hexadecimal conversion is same as octal.

For example:

(i) Convert  $(2598.675)_{10}$  into hexadecimal.

Solution:

|                                 | Remainder |     |                            |
|---------------------------------|-----------|-----|----------------------------|
|                                 | Decimal   | Hex | Hex                        |
| $16 \overline{) 2598}$          |           |     | $0.675 \times 16 = 10.8$ A |
| $16 \overline{) 162} \quad - 6$ | 6         | 6   | $0.800 \times 16 = 12.8$ C |
| $16 \overline{) 10} \quad - 2$  | 2         | 2   | $0.800 \times 16 = 12.8$ C |
| $0 \quad - 10$                  | A         |     | $0.800 \times 16 = 12.8$ C |

Result of  $(2598.675)_{10}$  is  $(A26.ACCC)_{16}$

### **3. OCTAL NUMBER SYSTEM:-**

#### **(a) Octal to binary conversion:-**

To convert a given a octal number to binary, replace each octal digit by its 3- bit binary equivalent.

**For example:**

**Convert  $(367.52)_8$  into binary.**

**Solution:**

Given Octal number is                    3    6    7    .    5    2  
Convert each group octal                = 011   110   111   .   101   010  
to binary

Result of  $(367.52)_8$  is  $(011110111.101010)_2$

**(b) Octal to decimal conversion:-**

For conversion octal to decimal number, multiply each digit in the octal number by the weight of its position and add all the product terms

**For example: -**

**Convert  $(4057.06)_8$  to decimal**

**Solution:**

$$\begin{aligned}(4057.06)_8 &= 4 \times 8^3 + 0 \times 8^2 + 5 \times 8^1 + 7 \times 8^0 + 0 \times 8^{-1} + 6 \times 8^{-2} \\ &= 2048 + 0 + 40 + 7 + 0 + 0.0937 \\ &= (2095.0937)_{10}\end{aligned}$$

Result is  $(2095.0937)_{10}$

**(c) Octal to hexadecimal conversion:-**

For conversion of octal to Hexadecimal, first convert the given octal number to binary and then binary number to hexadecimal.

**For example :-**

**Convert  $(756.603)_8$  to hexadecimal.**

**Solution :-**

|                                    |   |      |      |      |   |      |      |      |
|------------------------------------|---|------|------|------|---|------|------|------|
| Given octal no.                    |   | 7    | 5    | 6    | . | 6    | 0    | 3    |
| Convert each octal digit to binary | = | 111  | 101  | 110  | . | 110  | 000  | 011  |
| Group of 4bits are                 | = | 0001 | 1110 | 1110 | . | 1100 | 0001 | 1000 |
| Convert 4 bits group to hex.       | = | 1    | E    | E    | . | C    | 1    | 8    |

Result is  $(1EE.C18)_{16}$

**(4) HEXADECEMAL NUMBER SYSTEM :-**

**(a) Hexadecimal to binary conversion:-**

For conversion of hexadecimal to binary, replace hexadecimal digit by its 4 bit binary group.

**For example:**

**Convert  $(3A9E.B0D)_{16}$  into binary.**

**Solution:**

Given Hexadecimal number is            3    A    9    E    .    B    0    D  
Convert each hexadecimal                = 0011   1010   1001   1110   .   1011   0000   1101  
digit to 4 bit binary

Result of  $(3A9E.B0D)_{16}$  is  $(0011101010011110.101100001101)_2$

## **(b)Hexadecimal to decimal conversion:-**

For conversion of hexadecimal to decimal, multiply each digit in the hexadecimal number by its position weight and add all those product terms.

**For example: -**

**Convert (A0F9.0EB)<sub>16</sub> to decimal**

**Solution:**

$$\begin{aligned}(A0F9.0EB)_{16} &= (10 \times 16^3) + (0 \times 16^2) + (15 \times 16^1) + (9 \times 16^0) + (0 \times 16^{-1}) + (14 \times 16^{-2}) + (11 \times 16^{-3}) \\ &= 40960 + 0 + 240 + 9 + 0 + 0.0546 + 0.0026 \\ &= (41209.0572)_{10}\end{aligned}$$

Result is **(41209.0572)<sub>10</sub>**

## **(c) Hexadecimal to Octal conversion:-**

For conversion of hexadecimal to octal, first convert the given hexadecimal number to binary and then binary number to octal.

**For example :-**

**Convert (B9F.AE)<sub>16</sub> to octal.**

**Solution :-**

|                                   |   |      |      |      |     |      |      |     |     |
|-----------------------------------|---|------|------|------|-----|------|------|-----|-----|
| Given hexadecimal no.is           |   | B    | 9    | F    | .   | A    | E    |     |     |
| Convert each hex. digit to binary | = | 1011 | 1001 | 1111 | .   | 1010 | 1110 |     |     |
| Group of 3 bits are               | = | 101  | 110  | 011  | 111 | .    | 101  | 011 | 100 |
| Convert 3 bits group to octal.    | = | 5    | 6    | 3    | 7   | .    | 5    | 3   | 4   |

Result is **(5637.534)<sub>8</sub>**

## **BINARY ARITHMETIC OPERATION :-**

### **1. BINARY ADDITION:-**

The binary addition rules are as follows

$0 + 0 = 0$  ;  $0 + 1 = 1$  ;  $1 + 0 = 1$  ;  $1 + 1 = 10$  , i.e 0 with a carry of 1

**For example :-**

**Add (100101)<sub>2</sub> and (1101111)<sub>2</sub>.**

**Solution :-**

$$\begin{array}{r} 100101 \\ + \underline{1101111} \\ \hline 10010100 \end{array}$$

Result is **(10010100)<sub>2</sub>**

### **2. BINARY SUBTRACTION:-**

The binary subtraction rules are as follows

$0 - 0 = 0$  ;  $1 - 1 = 0$  ;  $1 - 0 = 1$  ;  $0 - 1 = 1$  , with a borrow of 1

For example :-

Subtract  $(111.111)_2$  from  $(1010.01)_2$ .

Solution :-

$$\begin{array}{r} 1010.010 \\ - 111.111 \\ \hline 0010.011 \end{array}$$

Result is  $(0010.011)_2$

### 3. BINARY MULTIPLICATION:-

The binary multiplication rules are as follows

$0 \times 0 = 0$  ;  $1 \times 1 = 1$  ;  $1 \times 0 = 0$  ;  $0 \times 1 = 0$

For example :-

Multiply  $(1101)_2$  by  $(110)_2$ .

Solution :-

$$\begin{array}{r} 1101 \\ x 110 \\ \hline 0000 \\ 1101 \\ + 1101 \\ \hline 1001110 \end{array}$$

Result is  $(1001110)_2$

### 4. BINARY DIVISION:-

The binary division is very simple and similar to decimal number system. The division by '0' is meaningless.

So we have only 2 rules

$0 \div 1 = 0$

$1 \div 1 = 1$

For example :-

Divide  $(10110)_2$  by  $(110)_2$ .

Solution :-

$$\begin{array}{r} 110 \ ) \ 101101 \ ( \ 111.1 \\ - \ 110 \\ \hline 1010 \\ \ 110 \\ \hline 1001 \\ \ 110 \\ \hline 110 \\ \ 110 \\ \hline 000 \end{array}$$

Result is  $(111.1)_2$

## 1's COMPLEMENT REPRESENTATION :-

The 1's complement of a binary number is obtained by changing each 0 to 1 and each 1 to 0.

**For example :-**

**Find  $(1100)_2$  1's complement.**

**Solution :-**

|                   |   |   |   |   |
|-------------------|---|---|---|---|
| Given             | 1 | 1 | 0 | 0 |
| 1's complement is | 0 | 0 | 1 | 1 |

Result is  $(0011)_2$

## 2's COMPLEMENT REPRESENTATION :-

The 2's complement of a binary number is a binary number which is obtained by adding 1 to the 1's complement of a number i.e.

$$2's \text{ complement} = 1's \text{ complement} + 1$$

**For example :-**

**Find  $(1010)_2$  2's complement.**

**Solution :-**

|                   |          |          |          |          |
|-------------------|----------|----------|----------|----------|
| Given             | 1        | 0        | 1        | 0        |
| 1's complement is | 0        | 1        | 0        | 1        |
|                   | +        |          |          | 1        |
| 2's complement    | <u>0</u> | <u>1</u> | <u>1</u> | <u>0</u> |

Result is  $(0110)_2$

## SIGNED NUMBER :-

In sign – magnitude form, additional bit called the sign bit is placed in front of the number. If the sign bit is 0, the number is positive. If it is a 1, the number is negative.

**For example:-**

|          |   |   |   |   |   |   |   |     |
|----------|---|---|---|---|---|---|---|-----|
| 0        | 1 | 0 | 1 | 0 | 0 | 1 | = | +41 |
| ↑        |   |   |   |   |   |   |   |     |
| Sign bit |   |   |   |   |   |   |   |     |
| 1        | 1 | 0 | 1 | 0 | 0 | 1 | = | -41 |
| ↑        |   |   |   |   |   |   |   |     |
| Sign bit |   |   |   |   |   |   |   |     |

## SUBTRACTION USING COMPLEMENT METHOD :-

### 1's COMPLEMENT:-

In 1's complement subtraction, add the 1's complement of subtrahend to the minuend. If there is a carry out, then the carry is added to the LSB. This is called end around carry. If the MSB is 0, the result is positive. If the MSB is 1, the result is negative and is in its 1's complement form. Then take its 1's complement to get the magnitude in binary.

For example:-

Subtract  $(10000)_2$  from  $(11010)_2$  using 1's complement.

Solution:-

$$\begin{array}{r} 11010 \\ - 10000 \\ \hline \end{array} \Rightarrow \begin{array}{r} 11010 \\ + \underline{01111} \text{ (1's complement)} \\ \hline \text{Carry} \rightarrow 101001 \\ + \underline{\phantom{1}1} \\ \hline 01010 = +10 \end{array} \begin{array}{l} = 26 \\ = -16 \\ + 10 \end{array}$$

Result is +10

**2's COMPLEMENT:-**

In 2's complement subtraction, add the 2's complement of subtrahend to the minuend. If there is a carry out, ignore it. If the MSB is 0, the result is positive. If the MSB is 1, the result is negative and is in its 2's complement form. Then take its 2's complement to get the magnitude in binary.

For example:-

Subtract  $(1010100)_2$  from  $(1010100)_2$  using 2's complement.

Solution:-

$$\begin{array}{r} 1010100 \\ - 1010100 \\ \hline \end{array} \Rightarrow \begin{array}{r} 1010100 \\ + \underline{0101100} \text{ (2's complement)} \\ \hline 10000000 \text{ (Ignore the carry)} \\ = 0 \text{ (result = 0)} \end{array} \begin{array}{l} = 84 \\ = -84 \\ \underline{0} \end{array}$$

Hence MSB is 0. The answer is positive. So it is +0000000 = 0

**DIGITAL CODES:-**

In practice the digital electronics requires to handle data which may be numeric, alphabets and special characters. This requires the conversion of the incoming data into binary format before it can be processed. There is various possible ways of doing this and this process is called encoding. To achieve the reverse of it, we use decoders.

**WEIGHTED AND NON-WEIGHTED CODES:-**

There are two types of binary codes

- 1) Weighted binary codes
- 2) Non-weighted binary codes

In weighted codes, for each position ( or bit ), there is specific weight attached.

For example, in binary number, each bit is assigned particular weight  $2^n$  where 'n' is the bit number for  $n = 0, 1, 2, 3, 4$  the weights are 1, 2, 4, 8, 16 respectively.

Example :- BCD

Non-weighted codes are codes which are not assigned with any weight to each digit position, i.e., each digit position within the number is not assigned fixed value.

Example:- Excess - 3 (XS -3) code and Gray codes

**BINARY CODED DECIMAL (BCD):-**

BCD is a weighted code. In weighted codes, each successive digit from right to left represents weights equal to some specified value and to get the equivalent decimal number add the products of the weights by the corresponding binary digit. 8421 is the most common because 8421 BCD is the most natural amongst the other possible codes.

**For example:-**

**(567)<sub>10</sub> is encoded in various 4 bit codes.**

**Solution:-**

|           |   |      |      |      |
|-----------|---|------|------|------|
| Decimal   | → | 5    | 6    | 7    |
| 8421 code | → | 0101 | 0110 | 0111 |
| 6311 code | → | 0111 | 1000 | 1001 |
| 5421 code | → | 1000 | 0100 | 1010 |

### **BCD ADDITION:-**

Addition of BCD (8421) is performed by adding two digits of binary, starting from least significant digit. In case if the result is an illegal code (greater than 9) or if there is a carry out of one then add 0110(6) and add the resulting carry to the next most significant.

**For example:-**

**Add 679.6 from 536.8 using BCD addition.**

**Solution:-**

|             |     |        |       |       |      |       |      |                           |
|-------------|-----|--------|-------|-------|------|-------|------|---------------------------|
| 6 7 9 . 6   |     | 0110   | 0111  | 1001  | .    | 0110  |      | ( 679.6 in BCD)           |
| + 5 3 6 . 8 | =>+ | 0101   | 0011  | 0110  | .    | 1000  |      | (536.8 in BCD)            |
| 1 2 1 6 . 4 |     | 1011   | 1010  | 1111  | .    | 1110  |      | ( All are illegal codes)  |
|             |     | + 0110 | +0110 | +0110 | .    | +0110 |      | ( Add 0110 to each)       |
|             |     | 0001   | 0010  | 0001  | 0110 | .     | 0100 |                           |
|             |     | 1      | 2     | 1     | 6    | .     | 4    | ( corrected sum = 1216.4) |

Result is **1216.4**

### **BCD SUBTRACTION:-**

The BCD subtraction is performed by subtracting the digits of each 4 – bit group of the subtrahend from corresponding 4 – bit group of the minuend in the binary starting from the LSD. If there is no borrow from the next higher group[ then no correction is required. If there is a borrow from the next group, then 6<sub>10</sub> (0110) is subtracted from the difference term of this group.

**For example:-**

**Subtract 147.8 from 206.7 using 8421 BCD code.**

**Solution:-**

|             |     |        |       |         |      |      |  |                                |
|-------------|-----|--------|-------|---------|------|------|--|--------------------------------|
| 2 0 6 . 7   |     | 0010   | 0000  | 0110    | .    | 0111 |  | ( 206.7 in BCD)                |
| - 1 4 7 . 8 | =>- | 0001   | 0100  | 0111    | .    | 1000 |  | (147.8 in BCD)                 |
| 5 8 . 9     |     | 0000   | 1011  | 1110    | .    | 1111 |  | ( Borrows are present)         |
|             |     | - 0110 | -0110 | .- 0110 |      |      |  |                                |
|             |     | 0101   | 1000  | .       | 1001 |      |  |                                |
|             |     | 5      | 8     | .       | 9    |      |  | ( corrected difference = 58.9) |

Result is **(58.9)<sub>10</sub>**

### **EXCESS THREE(XS-3) CODE:-**

The Excess-3 code, also called XS-3, is a non- weighted BCD code. This derives it name from the fact that each binary code word is the corresponding 8421 code word plus 0011(3). It is a sequential code. It is a self complementing code.

### XS-3 ADDITION:-

In XS-3 addition, add the XS-3 numbers by adding the 4 bit groups in each column starting from the LSD. If there is no carry out from the addition of any of the 4 bit groups, subtract 0011 from the sum term of those groups. If there is a carry out, add 0011 to the sum term of those groups

**For example:-**

**Add 37 and 28 using XS-3 code.**

**Solution:-**

$$\begin{array}{r}
 37 \quad \quad \quad 0110 \ 1010 \quad (37 \text{ in XS-3}) \\
 + \underline{28} \quad \Rightarrow \quad + \underline{0101 \ 1011} \quad (28 \text{ in XS-3}) \\
 \hline
 65 \quad \quad \quad 1011 \ 11010 \quad (\text{Carry is generated}) \\
 \quad \quad \quad + \underline{\quad 1} \quad \quad \quad (\text{Propagate carry}) \\
 \quad \quad \quad 1100 \ 0101 \quad (\text{Add } 0110 \text{ to correct } 0101 \text{ and} \\
 \quad \quad \quad - \underline{0011} \ +0011 \quad \text{subtract } 0011 \text{ to correct } 1100) \\
 \quad \quad \quad 1001 \ 1000 \quad (\text{Corrected sum in XS-3} = 65_{10})
 \end{array}$$

### XS-3 SUBTRACTION:-

To subtract in XS-3 number by subtracting each 4-bit group of the subtrahend from the corresponding 4-bit group of the minuend starting from the LSD. If there is no borrow from the next 4-bit group. add 0011 to the difference term of such groups. If there is a borrow, subtract 0011 from the difference term.

**For example :-**

**Subtract 175 from 267 using XS-3 code.**

**Solution :-`**

$$\begin{array}{r}
 267 \quad \quad \quad 0101 \ 1010 \ 1010 \quad (267 \text{ in XS-3}) \\
 - \underline{175} \quad \Rightarrow \quad - \underline{0100 \ 1010 \ 1000} \quad (175 \text{ in XS-3}) \\
 \hline
 092 \quad \quad \quad 0000 \ 1111 \ 0010 \quad (\text{Correct } 0010 \text{ and } 0000 \text{ by adding } 0011 \text{ and} \\
 \quad \quad \quad + \underline{0011} \ -0011 \ +0011 \quad \text{correct } 1111 \text{ by subtracting } 0011) \\
 \quad \quad \quad 0011 \ 1100 \ 0101 \quad (\text{Corrected difference in XS-3} = 92_{10})
 \end{array}$$

### ASCII CODE:-

The American Standard Code for Information Interchange (ASCII) pronounced as 'ASKEE' is widely used alphanumeric code. This is basically a 7 bit code. The number of different bit patterns that can be created with 7 bits is  $2^7 = 128$ , the ASCII can be used to encode both the uppercase and lowercase characters of the alphabet (52 symbols) and some special symbols in addition to the 10 decimal digits. It is used extensively for printers and terminals that interface with small computer systems. The table shown below shows the ASCII groups.

**The ASCII code**

| LSBs | MSBs |     |       |     |     |     |     |     |
|------|------|-----|-------|-----|-----|-----|-----|-----|
|      | 000  | 001 | 010   | 011 | 100 | 101 | 110 | 111 |
| 0000 | NUL  | DEL | Space | 0   | @   | P   | P   |     |
| 0001 | SOH  | DC1 | !     | 1   | A   | Q   | a   | q   |
| 0010 | STX  | DC2 | "     | 2   | B   | R   | b   | r   |
| 0011 | ETX  | DC3 | #     | 3   | C   | S   | c   | s   |



## GRAY CODE:-

The gray code is a non-weighted code. It is not a BCD code. It is cyclic code because successive words in this differ in one bit position only i.e it is a unit distance code.

Gray code is used in instrumentation and data acquisition systems where linear or angular displacement is measured. They are also used in shaft encoders, I/O devices, A/D converters and other peripheral equipment.

### BINARY- TO – GRAY CONVERSION:-

If an n-bit binary number is represented by  $B_n B_{n-1} \dots B_1$  and its gray code equivalent by  $G_n G_{n-1} \dots G_1$ , where  $B_n$  and  $G_n$  are the MSBs, then gray code bits are obtained from the binary code as follows

$$\begin{aligned} G_n &= B_n \\ G_{n-1} &= B_n \oplus B_{n-1} \\ &\vdots \\ &\vdots \\ &\vdots \\ G_1 &= B_2 \oplus B_1 \end{aligned}$$

Where the symbol  $\oplus$  stands for Exclusive OR (X-OR)

**For example :-**

**Convert the binary 1001 to the Gray code.**

**Solution :-`**

$$\begin{array}{cccc} \text{Binary} & \rightarrow & 1 & \text{--- } \oplus \text{---} & 0 & \text{--- } \oplus \text{---} & 0 & \text{--- } \oplus \text{---} & 1 \\ & & \downarrow & & \downarrow & & \downarrow & & \downarrow \\ \text{Gray} & \rightarrow & 1 & & 1 & & 0 & & 1 \end{array}$$

The gray code is **1101**

### GRAY- TO - BINARY CONVERSION:-

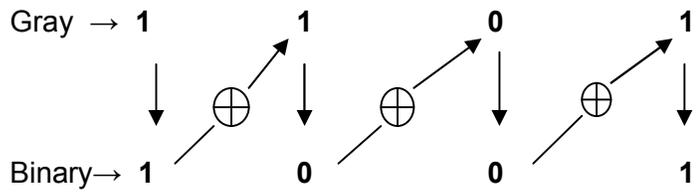
If an n-bit gray number is represented by  $G_n G_{n-1} \dots G_1$  and its binary equivalent by  $B_n B_{n-1} \dots B_1$ , then binary bits are obtained from Gray bits as follows :

$$\begin{aligned} B_n &= G_n \\ B_{n-1} &= B_n \oplus G_{n-1} \\ &\vdots \\ &\vdots \\ &\vdots \\ B_1 &= B_2 \oplus G_1 \end{aligned}$$

For example :-

Convert the Gray code 1101 to the binary.

Solution :-



The binary code is **1001**

# LOGIC GATES

## LOGIC GATES:-

- Logic gates are the fundamental building blocks of digital systems.
- There are 3 basic types of gates AND, OR and NOT.
- Logic gates are electronic circuits because they are made up of a number of electronic devices and components.
- Inputs and outputs of logic gates can occur only in 2 levels. These two levels are termed HIGH and LOW, or TRUE and FALSE, or ON and OFF or simply 1 and 0.
- The table which lists all the possible combinations of input variables and the corresponding outputs is called a truth table.

## LEVEL LOGIC:-

A logic in which the voltage levels represents logic 1 and logic 0. Level logic may be positive or negative logic.

### **Positive Logic:-**

A positive logic system is the one in which the higher of the two voltage levels represents the logic 1 and the lower of the two voltages level represents the logic 0.

### **Negative Logic:-**

A negative logic system is the one in which the lower of the two voltage levels represents the logic 1 and the higher of the two voltages level represents the logic 0.

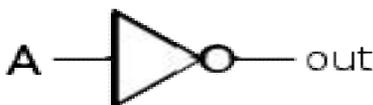
## DIFFERENT TYPES OF LOGIC GATES:-

### NOT GATE (INVERTER):-

- A NOT gate, also called an inverter, has only one input and one output.
- It is a device whose output is always the complement of its input.
- The output of a NOT gate is the logic 1 state when its input is in logic 0 state and the logic 0 state when its inputs is in logic 1 state.

IC No. :- 7404

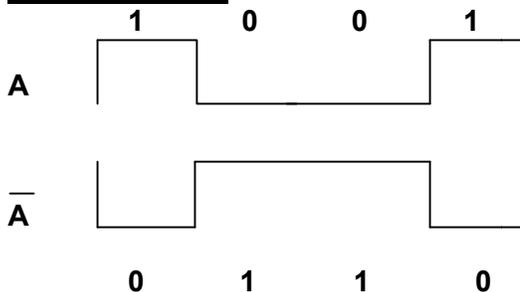
### Logic Symbol



### Truth table

| INPUT<br>A | OUTPUT<br>$\bar{A}$ |
|------------|---------------------|
| 0          | 1                   |
| 1          | 0                   |

### Timing Diagram

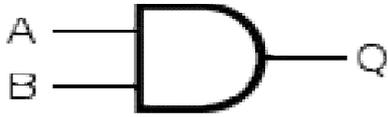


## AND GATE:-

- An AND gate has two or more inputs but only one output.
- The output is logic 1 state only when each one of its inputs is at logic 1 state.
- The output is logic 0 state even if one of its inputs is at logic 0 state.

IC No.:- 7408

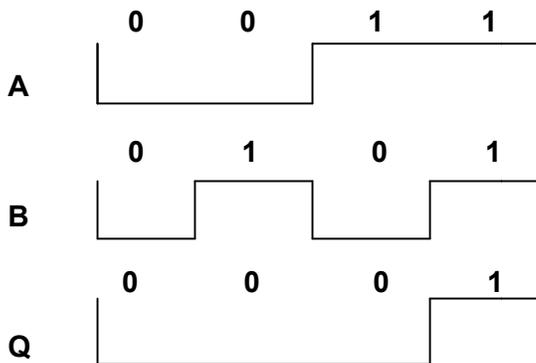
### Logic Symbol



### Truth Table

| INPUT |   | OUTPUT        |
|-------|---|---------------|
| A     | B | $Q=A \cdot B$ |
| 0     | 0 | 0             |
| 0     | 1 | 0             |
| 1     | 0 | 0             |
| 1     | 1 | 1             |

### Timing Diagram



## OR GATE:-

- An OR gate may have two or more inputs but only one output.
- The output is logic 1 state, even if one of its input is in logic 1 state.
- The output is logic 0 state, only when each one of its inputs is in logic state.

IC No.:- 7432

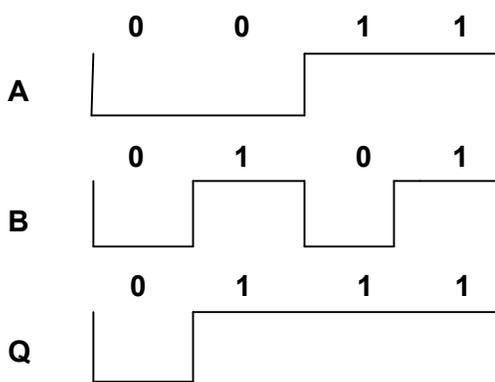
### Logic Symbol



### Truth Table

| INPUT |   | OUTPUT    |
|-------|---|-----------|
| A     | B | $Q=A + B$ |
| 0     | 0 | 0         |
| 0     | 1 | 1         |
| 1     | 0 | 1         |
| 1     | 1 | 1         |

### Timing Diagram



## NAND GATE:-

- NAND gate is a combination of an AND gate and a NOT gate.
- The output is logic 0 when each of the input is logic 1 and for any other combination of inputs, the output is logic 1.

IC No.:- 7400 two input NAND gate  
7410 three input NAND gate  
7420 four input NAND gate  
7430 eight input NAND gate

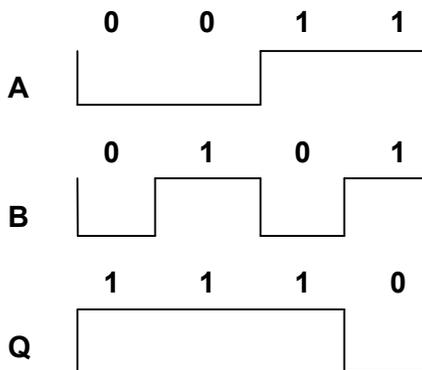
### Logic Symbol



### Truth Table

| INPUT |   | OUTPUT                     |
|-------|---|----------------------------|
| A     | B | $Q = \overline{A \cdot B}$ |
| 0     | 0 | 1                          |
| 0     | 1 | 1                          |
| 1     | 0 | 1                          |
| 1     | 1 | 0                          |

### Timing Diagram



## NOR GATE:-

- NOR gate is a combination of an OR gate and a NOT gate.
- The output is logic 1, only when each one of its input is logic 0 and for any other combination of inputs, the output is a logic 0 level.

IC No.:- 7402 two input NOR gate  
7427 three input NOR gate  
7425 four input NOR gate

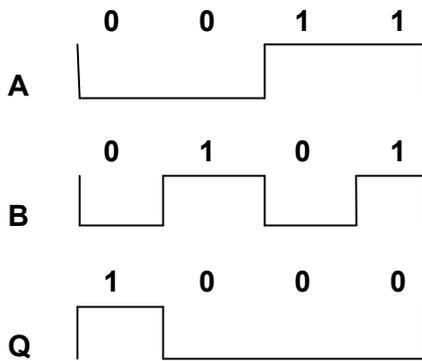
### Logic Symbol



### Truth Table

| INPUT |   | OUTPUT                 |
|-------|---|------------------------|
| A     | B | $Q = \overline{A + B}$ |
| 0     | 0 | 1                      |
| 0     | 1 | 0                      |
| 1     | 0 | 0                      |
| 1     | 1 | 0                      |

### Timing Diagram



### EXCLUSIVE – OR (X-OR) GATE:-

- An X-OR gate is a two input, one output logic circuit.
- The output is logic 1 when one and only one of its two inputs is logic 1. When both the inputs is logic 0 or when both the inputs is logic 1, the output is logic 0.

IC No.:- 7486

### Logic Symbol



INPUTS are **A** and **B**

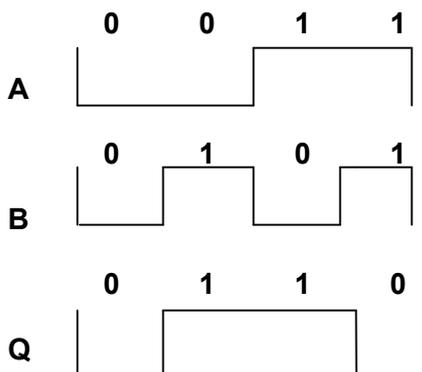
OUTPUT is  $Q = A \oplus B$

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

### Truth Table

| INPUT |   | OUTPUT           |
|-------|---|------------------|
| A     | B | $Q = A \oplus B$ |
| 0     | 0 | 0                |
| 0     | 1 | 1                |
| 1     | 0 | 1                |
| 1     | 1 | 0                |

### Timing Diagram

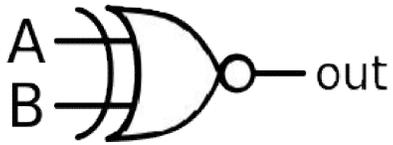


### EXCLUSIVE – NOR (X-NOR) GATE:-

- An X-NOR gate is the combination of an X-OR gate and a NOT gate.
- An X-NOR gate is a two input, one output logic circuit.
- The output is logic 1 only when both the inputs are logic 0 or when both the inputs is 1.
- The output is logic 0 when one of the inputs is logic 0 and other is 1.

IC No.:- 74266

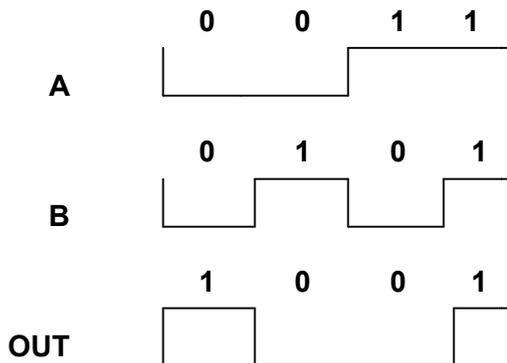
Logic Symbol



| INPUT |   | OUTPUT         |
|-------|---|----------------|
| A     | B | OUT = A XNOR B |
| 0     | 0 | 1              |
| 0     | 1 | 0              |
| 1     | 0 | 0              |
| 1     | 1 | 1              |

$$\begin{aligned} \text{OUT} &= A B + \bar{A} \bar{B} \\ &= A \text{ XNOR } B \end{aligned}$$

Timing Diagram

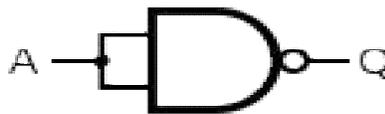


### UNIVERSAL GATES:-

There are 3 basic gates AND, OR and NOT, there are two universal gates NAND and NOR, each of which can realize logic circuits single handedly. The NAND and NOR gates are called universal building blocks. Both NAND and NOR gates can perform all logic functions i.e. AND, OR, NOT, EXOR and EXNOR.

#### NAND GATE:-

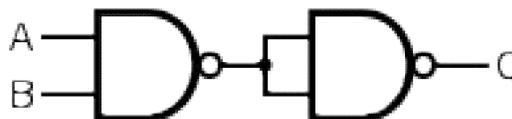
a) Inverter from NAND gate



$$\begin{aligned} \text{Input} &= A \\ \text{Output } Q &= \bar{A} \end{aligned}$$

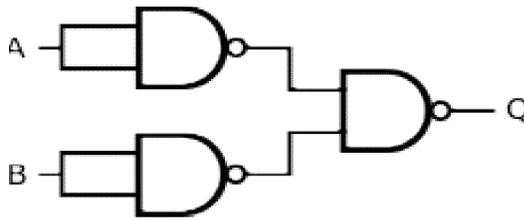
b) AND gate from NAND gate

Input s are A and B  
Output Q = A.B



c) OR gate from NAND gate

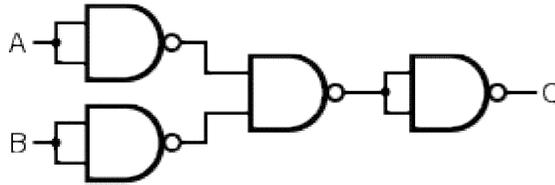
Inputs are A and B  
Output Q = A+B



**d) NOR gate from NAND gate**

Inputs are A and B

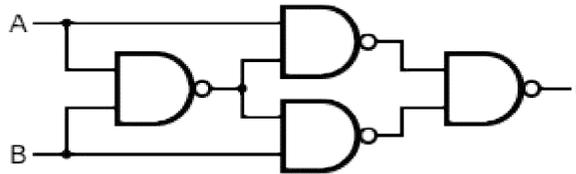
Output  $Q = \overline{A+B}$



**e) EX-OR gate from NAND gate**

Inputs are A and B

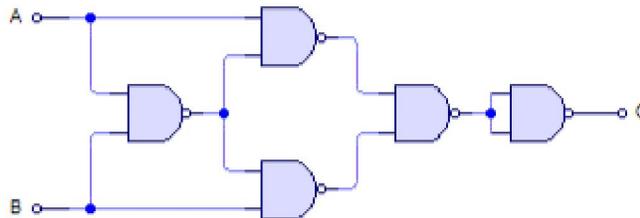
Output  $Q = A\overline{B} + \overline{A}B$



**f) EX-NOR gate From NAND gate**

Inputs are A and B

Output  $Q = \overline{A\overline{B} + \overline{A}B}$



**NOR GATE:-**

**a) Inverter from NOR gate**

Input = A

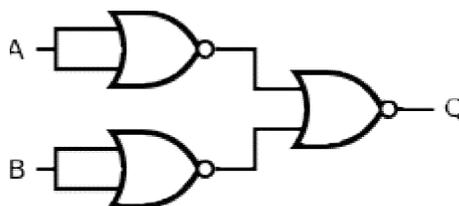
Output  $Q = \overline{A}$



**b) AND gate from NOR gate**

Input s are A and B

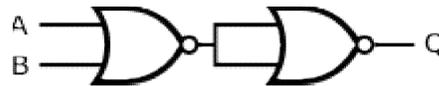
Output  $Q = A.B$



c) **OR gate from NOR gate**

Inputs are **A** and **B**

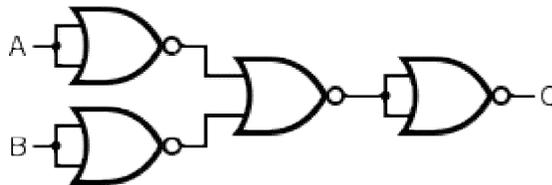
Output **Q = A+B**



d) **NAND gate from NOR gate**

Inputs are **A** and **B**

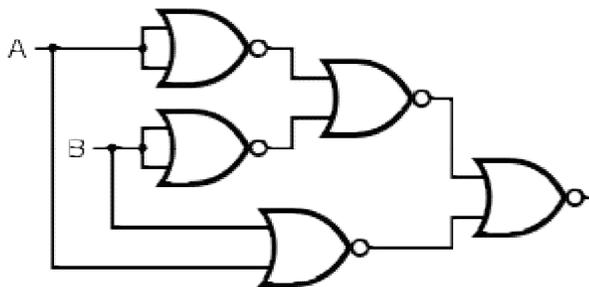
Output **Q =  $\overline{A \cdot B}$**



e) **EX-OR gate from NOR gate**

Inputs are **A** and **B**

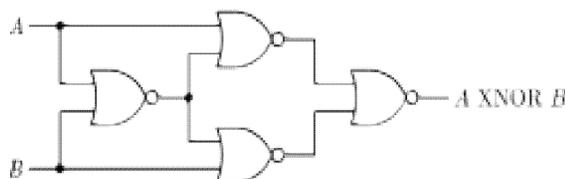
Output **Q =  $A\overline{B} + \overline{A}B$**



f) **EX-NOR gate From NOR gate**

Inputs are **A** and **B**

Output **Q =  $\overline{A\overline{B} + \overline{A}B}$**



## **THRESHOLD LOGIC:-**

### **INTRODUCTION:-**

- The threshold element, also called the threshold gate (T-gate) is a much more powerful device than any of the conventional logic gates such as NAND, NOR and others.
- Complex, large Boolean functions can be realized using much fewer threshold gates.
- Frequently a single threshold gate can realize a very complex function which otherwise might require a large number of conventional gates.
- T-gate offers incomparably economical realization; it has not found extensive use with the digital system designers mainly because of the following limitations.
  1. It is very sensitive to parameter variations.
  2. It is difficult to fabricate it in IC form.

3. The speed of switching of threshold elements is much lower than that of conventional gates.

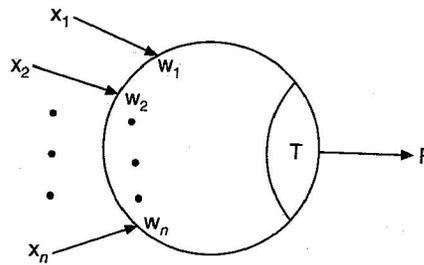
**THE THRESHOLD ELEMENTS:-**

- A threshold element or gate has 'n' binary inputs  $x_1, x_2, \dots, x_n$ ; and a single binary output F. But in addition to those, it has two more parameters.
- Its parameters are a threshold T and weights  $w_1, w_2, \dots, w_n$ . The weights  $w_1, w_2, \dots, w_n$  are associated with the input variables  $x_1, x_2, \dots, x_n$ .
- The value of the threshold (T) and weights may be real, positive or negative number.
- The symbol of the threshold element is shown in fig.(a).
- It is represented by a circle partitioned into two parts, one part represents the weights and other represents T.
- It is defined as

$$F(x_1, x_2, \dots, x_n) = 1 \text{ if and only if } \sum_{i=1}^n w_i x_i \geq T$$

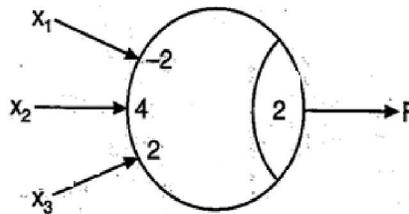
otherwise  
 $F(x_1, x_2, \dots, x_n) = 0$

- The sum and product operation are normal arithmetic operations and the sum  $\sum_{i=1}^n w_i x_i \geq T$  is called the weighted sum of the element or gate.



**Example:-**

Obtain the minimal Boolean expression from the threshold gate shown in figure.



**Solution:-**

The threshold gate with three inputs  $x_1, x_2, x_3$  with weights  $-2(w_1)$ ,  $4(w_2)$  and  $2(w_3)$  respectively. The value of threshold is 2(T). The table shown is the weighted sums and outputs for all input combinations. For this threshold gate, the weighted sum is

$$\begin{aligned} W &= w_1x_1 + w_2x_2 + w_3x_3 \\ &= (-2)x_1 + (4)x_2 + (2)x_3 \\ &= -2x_1 + 4x_2 + 2x_3 \end{aligned}$$

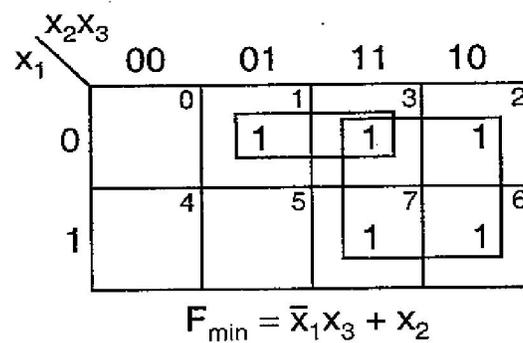
The output F is logic 1 for  $w \geq 2$  and it is logic 0 for  $w < 2$

| Input Variables |       |       | Weighted Sum              | Output |
|-----------------|-------|-------|---------------------------|--------|
| $x_1$           | $x_2$ | $x_3$ | $w = -2x_1 + 4x_2 + 2x_3$ | F      |
| 0               | 0     | 0     | 0                         | 0      |
| 0               | 0     | 1     | 2                         | 1      |
| 0               | 1     | 0     | 4                         | 1      |
| 0               | 1     | 1     | 6                         | 1      |
| 1               | 0     | 0     | -2                        | 0      |
| 1               | 0     | 1     | 0                         | 0      |
| 1               | 1     | 0     | 2                         | 1      |
| 1               | 1     | 1     | 4                         | 1      |

From the input – output relation is given in the table, the Boolean expression for the output is

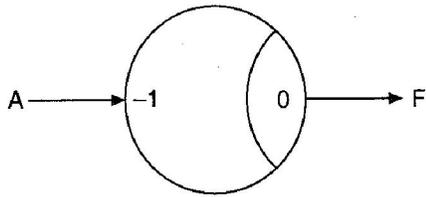
$$F = \sum m(1, 2, 3, 6, 7)$$

The K-map for F is



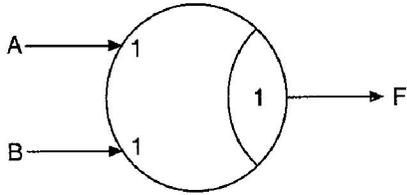
### UNIVERSALITY OF A T-GATE:-

- A single T-gate can realize a large number of functions by merely changing either the weights or the threshold or both, which can be done by altering the value of the corresponding resistors.
- Since a threshold gate can realize universal gates, i.e., NAND gates and NOR gates, a threshold gate is also a universal gate.
- Single threshold gate cannot realize by a single T-gate
- Realization of logic gates using T-gates is shown in the below figure.



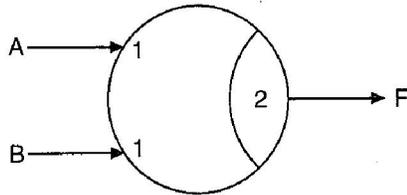
(a) NOT gate  $F = \bar{A}$

| Input | Weighted sum | Output |
|-------|--------------|--------|
| A     | $W = -A$     | F      |
| 0     | 0            | 1      |
| 1     | -1           | 0      |



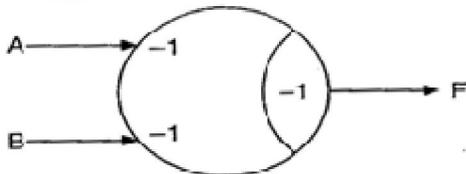
(b) OR gate  $F = A + B$

| Inputs | Weighted sum | Output |
|--------|--------------|--------|
| A B    | $w = A + B$  | F      |
| 0 0    | 0            | 0      |
| 0 1    | 1            | 1      |
| 1 0    | 1            | 1      |
| 1 1    | 2            | 1      |



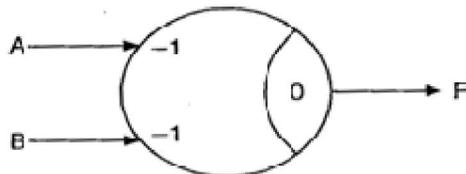
(c) AND gate  $F = AB$

| Inputs | Weighted sum | Output |
|--------|--------------|--------|
| A B    | $w = A + B$  | F      |
| 0 0    | 0            | 0      |
| 0 1    | 1            | 0      |
| 1 0    | 1            | 0      |
| 1 1    | 2            | 1      |



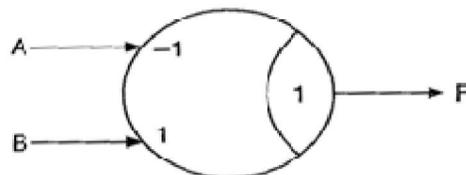
NAND gate  
(d)  $F = \overline{A + B} = \bar{A}\bar{B}$

| Inputs | Weighted sum | Output |
|--------|--------------|--------|
| A B    | $w = -A - B$ | F      |
| 0 0    | 0            | 1      |
| 0 1    | -1           | 1      |
| 1 0    | -1           | 1      |
| 1 1    | -2           | 0      |



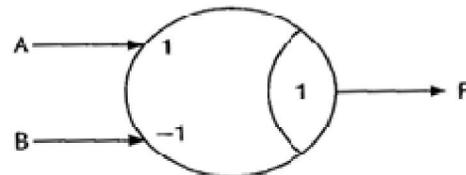
NOR gate  
(e)  $F = \overline{A \cdot B} = \bar{A} + \bar{B}$

| Inputs | Weighted sum | Output |
|--------|--------------|--------|
| A B    | $w = -A - B$ | F      |
| 0 0    | 0            | 1      |
| 0 1    | -1           | 0      |
| 1 0    | -1           | 0      |
| 1 1    | -2           | 0      |



(f)  $F = \overline{AB}$

| Inputs | Weighted sum | Output |
|--------|--------------|--------|
| A B    | $w = -A + B$ | F      |
| 0 0    | 0            | 0      |
| 0 1    | 1            | 1      |
| 1 0    | -1           | 0      |
| 1 1    | 0            | 0      |



(g)  $F = A\bar{B}$

| Inputs | Weighted sum | Output |
|--------|--------------|--------|
| A B    | $w = A - B$  | F      |
| 0 0    | 0            | 0      |
| 0 1    | -1           | 0      |
| 1 0    | 1            | 1      |
| 1 1    | 0            | 0      |

# BOOLEAN ALGEBRA

## INTRODUCTION:-

- Switching circuits are also called logic circuits, gates circuits and digital circuits.
- Switching algebra is also called Boolean algebra.
- Boolean algebra is a system of mathematical logic. It is an algebraic system consisting of the set of elements (0,1), two binary operators called OR and AND and unary operator called NOT.
- It is the basic mathematical tool in the analysis and synthesis of switching circuits.
- It is a way to express logic functions algebraically.
- Any complex logic can be expressed by a Boolean function.
- The Boolean algebra is governed by certain well developed rules and laws.

## AXIOMS AND LAWS OF BOOLEAN ALGEBRA:-

Axioms or postulates of Boolean algebra are set of logical expressions that are accepted without proof and upon which we can build a set of useful theorems. Actually, axioms are nothing more than the definitions of the three basic logic operations AND, OR and INVERTER. Each axiom can be interpreted as the outcome of an operation performed by a logic gate.

### **AND operation**

Axiom 1:  $0 \cdot 0 = 0$

Axiom 2:  $0 \cdot 1 = 0$

Axiom 3:  $1 \cdot 0 = 0$

Axiom 4:  $1 \cdot 1 = 1$

### **OR operation**

Axiom 5:  $0 + 0 = 0$

Axiom 6:  $0 + 1 = 1$

Axiom 7:  $1 + 0 = 1$

Axiom 8:  $1 + 1 = 1$

### **NOT operation**

Axiom 9:  $\overline{\overline{1}} = 0$

Axiom 10:  $\overline{\overline{0}} = 1$

### **1. Complementation Laws:-**

The term complement simply means to invert, i.e. to changes 0s to 1s and 1s to 0s. The five laws of complementation are as follows:

**Law 1:**  $\overline{0} = 1$

**Law 2:**  $\overline{1} = 0$

**Law 3:** if  $A = 0$ , then  $\overline{\overline{A}} = 1$

**Law 4:** if  $\overline{A} = 1$ , then  $\overline{\overline{A}} = 0$

**Law 5:**  $\overline{\overline{A}} = A$  (double complementation law)

### **2. OR Laws:-**

The four OR laws are as follows

**Law 1:**  $A + 0 = A$  (Null law)

**Law 2:**  $A + 1 = 1$  (Identity law)

**Law 3:**  $A + A = A$

**Law 4:**  $A + \overline{A} = 1$

### **3. AND Laws:-**

The four AND laws are as follows

**Law 1:**  $A \cdot 0 = 0$  (Null law)

**Law 2:**  $A \cdot 1 = A$  (Identity law)

**Law 3:**  $A \cdot A = A$

**Law 4:**  $A \cdot \overline{A} = 0$

#### 4. Commutative Laws:-

Commutative laws allow change in position of AND or OR variables. There are two commutative laws.

**Law 1:**  $A + B = B + A$

**Proof**

| A | B | A + B |
|---|---|-------|
| 0 | 0 | 0     |
| 0 | 1 | 1     |
| 1 | 0 | 1     |
| 1 | 1 | 1     |

=

| B | A | B + A |
|---|---|-------|
| 0 | 0 | 0     |
| 0 | 1 | 1     |
| 1 | 0 | 1     |
| 1 | 1 | 1     |

**Law 2:**  $A \cdot B = B \cdot A$

**Proof**

| A | B | A . B |
|---|---|-------|
| 0 | 0 | 0     |
| 0 | 1 | 0     |
| 1 | 0 | 0     |
| 1 | 1 | 1     |

=

| B | A | B . A |
|---|---|-------|
| 0 | 0 | 0     |
| 0 | 1 | 0     |
| 1 | 0 | 0     |
| 1 | 1 | 1     |

This law can be extended to any number of variables. For example

$A \cdot B \cdot C = B \cdot C \cdot A = C \cdot A \cdot B = B \cdot A \cdot C$

#### 5. Associative Laws:-

The associative laws allow grouping of variables. There are 2 associative laws.

**Law 1:**  $(A + B) + C = A + (B + C)$

**Proof**

| A | B | C | A+B | (A+B)+C |
|---|---|---|-----|---------|
| 0 | 0 | 0 | 0   | 0       |
| 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   | 1       |

=

| A | B | C | B+C | A+(B+C) |
|---|---|---|-----|---------|
| 0 | 0 | 0 | 0   | 0       |
| 0 | 0 | 1 | 1   | 1       |
| 0 | 1 | 0 | 1   | 1       |
| 0 | 1 | 1 | 1   | 1       |
| 1 | 0 | 0 | 0   | 1       |
| 1 | 0 | 1 | 1   | 1       |
| 1 | 1 | 0 | 1   | 1       |
| 1 | 1 | 1 | 1   | 1       |

**Law 2:**  $(A \cdot B) \cdot C = A \cdot (B \cdot C)$

**Proof**

| A | B | C | AB | (AB)C |
|---|---|---|----|-------|
| 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 | 1  | 0     |
| 1 | 1 | 1 | 1  | 1     |

=

| A | B | C | B.C | A(B.C) |
|---|---|---|-----|--------|
| 0 | 0 | 0 | 0   | 0      |
| 0 | 0 | 1 | 0   | 0      |
| 0 | 1 | 0 | 0   | 0      |
| 0 | 1 | 1 | 1   | 0      |
| 1 | 0 | 0 | 0   | 0      |
| 1 | 0 | 1 | 0   | 0      |
| 1 | 1 | 0 | 0   | 0      |
| 1 | 1 | 1 | 1   | 1      |

This law can be extended to any number of variables. For example

$$A(BCD) = (ABC)D = (AB)(CD)$$

**6. Distributive Laws:-**

The distributive laws allow factoring or multiplying out of expressions. There are two distributive laws.

**Law 1:**  $A(B + C) = AB + AC$

**Proof**

| A | B | C | B+C | A(B+C) |
|---|---|---|-----|--------|
| 0 | 0 | 0 | 0   | 0      |
| 0 | 0 | 1 | 1   | 0      |
| 0 | 1 | 0 | 1   | 0      |
| 0 | 1 | 1 | 1   | 0      |
| 1 | 0 | 0 | 0   | 0      |
| 1 | 0 | 1 | 1   | 1      |
| 1 | 1 | 0 | 1   | 1      |
| 1 | 1 | 1 | 1   | 1      |

=

| A | B | C | AB | AC | A+(B+C) |
|---|---|---|----|----|---------|
| 0 | 0 | 0 | 0  | 0  | 0       |
| 0 | 0 | 1 | 0  | 0  | 0       |
| 0 | 1 | 0 | 0  | 0  | 0       |
| 0 | 1 | 1 | 0  | 0  | 0       |
| 1 | 0 | 0 | 0  | 0  | 0       |
| 1 | 0 | 1 | 0  | 1  | 1       |
| 1 | 1 | 0 | 1  | 0  | 1       |
| 1 | 1 | 1 | 1  | 1  | 1       |

**Law 2:**  $A + BC = (A+B)(A+C)$

**Proof**

$$\begin{aligned} \text{RHS} &= (A+B)(A+C) \\ &= AA + AC + BA + BC \\ &= A + AC + AB + BC \\ &= A(1 + C + B) + BC \\ &= A \cdot 1 + BC \\ &= A + BC \\ &= \text{LHS} \end{aligned}$$

$$(1 + C + B = 1 + B = 1)$$

**7. Redundant Literal Rule (RLR):-**

**Law 1:**  $A + \bar{A}B = A + B$

**Proof**

$$\begin{aligned}
 A + \bar{A}B &= (A + \bar{A})(A + B) \\
 &= 1 \cdot (A + B) \\
 &= A + B
 \end{aligned}$$

**Law 2:**  $A(\bar{A} + B) = AB$

**Proof**

$$\begin{aligned}
 A(\bar{A} + B) &= A\bar{A} + AB \\
 &= 0 + AB \\
 &= AB
 \end{aligned}$$

**8. Idempotence Laws:-**

Idempotence means same value.

**Law 1:**  $A \cdot A = A$

**Proof**

If  $A = 0$ , then  $A \cdot A = 0 \cdot 0 = 0 = A$

If  $A = 1$ , then  $A \cdot A = 1 \cdot 1 = 1 = A$

This law states that AND of a variable with itself is equal to that variable only.

**Law 2:**  $A + A = A$

**Proof**

If  $A = 0$ , then  $A + A = 0 + 0 = 0 = A$

If  $A = 1$ , then  $A + A = 1 + 1 = 1 = A$

This law states that OR of a variable with itself is equal to that variable only.

**9. Absorption Laws:-**

There are two laws:

**Law 1:**  $A + A \cdot B = A$

**Proof**

$$A + A \cdot B = A(1 + B) = A \cdot 1 = A$$

| A | B | AB | A+AB |
|---|---|----|------|
| 0 | 0 | 0  | 0    |
| 0 | 1 | 0  | 0    |
| 1 | 0 | 0  | 1    |
| 1 | 1 | 1  | 1    |

**Law 2:**  $A(A + B) = A$

**Proof**

$$A(A + B) = A \cdot A + A \cdot B = A + AB = A(1 + B) = A \cdot 1 = A$$

| A | B | A+B | A(A+B) |
|---|---|-----|--------|
| 0 | 0 | 0   | 0      |
| 0 | 1 | 1   | 0      |
| 1 | 0 | 1   | 1      |
| 1 | 1 | 1   | 1      |

## 10. Consensus Theorem (Included Factor Theorem):-

### Theorem 1:

$$AB + \overline{A}C + BC = AB + \overline{A}C$$

### Proof

$$\begin{aligned} \text{LHS} &= AB + \overline{A}C + BC \\ &= AB + \overline{A}C + BC(A + \overline{A}) \\ &= AB + \overline{A}C + BCA + BC\overline{A} \\ &= AB(1 + C) + \overline{A}C(1 + B) \\ &= AB(1) + \overline{A}C(1) \\ &= AB + \overline{A}C \\ &= \text{RHS} \end{aligned}$$

### Theorem 2:

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

### Proof

$$\begin{aligned} \text{LHS} &= (A + B)(\overline{A} + C)(B + C) \\ &= (A\overline{A} + AC + B\overline{A} + BC)(B + C) \\ &= (AC + BC + \overline{A}B)(B + C) \\ &= ABC + BC + \overline{A}B + AC + BC + \overline{A}BC \\ &= AC + BC + \overline{A}B \\ \text{RHS} &= (A + B)(\overline{A} + C) \\ &= A\overline{A} + AC + BC + \overline{A}B \\ &= AC + BC + \overline{A}B \\ &= \text{LHS} \end{aligned}$$

## 11. Transposition Theorem:-

### Theorem:

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

### Proof

$$\begin{aligned} \text{RHS} &= (A + C)(\overline{A} + B) \\ &= A\overline{A} + C\overline{A} + AB + CB \\ &= 0 + \overline{A}C + AB + BC \\ &= \overline{A}C + AB + BC(A + \overline{A}) \\ &= \overline{A}C + AB + \overline{A}BC + \overline{A}BC \\ &= \overline{A}C + AB \\ &= \text{LHS} \end{aligned}$$

## 12. De Morgan's Theorem:-

De Morgan's theorem represents two laws in Boolean algebra.

**Law 1:**  $\overline{A + B} = \overline{A} \cdot \overline{B}$

### Proof

| A | B | A + B | $\overline{A + B}$ |
|---|---|-------|--------------------|
| 0 | 0 | 0     | 1                  |
| 0 | 1 | 1     | 0                  |
| 1 | 0 | 1     | 0                  |
| 1 | 1 | 1     | 0                  |

=

| A | B | $\overline{A}$ | $\overline{B}$ | $\overline{A} \cdot \overline{B}$ |
|---|---|----------------|----------------|-----------------------------------|
| 0 | 0 | 1              | 1              | 1                                 |
| 0 | 1 | 1              | 0              | 0                                 |
| 1 | 0 | 0              | 1              | 0                                 |
| 1 | 1 | 0              | 0              | 0                                 |

This law states that the complement of a sum of variables is equal to the product of their individual complements.

**Law 2:**  $\overline{A \cdot B} = \overline{A} + \overline{B}$

**Proof**

| A | B | A · B | $\overline{A \cdot B}$ |
|---|---|-------|------------------------|
| 0 | 0 | 0     | 1                      |
| 0 | 1 | 0     | 1                      |
| 1 | 0 | 0     | 1                      |
| 1 | 1 | 1     | 0                      |

=

| A | B | $\overline{A}$ | $\overline{B}$ | $\overline{A + B}$ |
|---|---|----------------|----------------|--------------------|
| 0 | 0 | 1              | 1              | 1                  |
| 0 | 1 | 1              | 0              | 1                  |
| 1 | 0 | 0              | 1              | 1                  |
| 1 | 1 | 0              | 0              | 0                  |

This law states that the complement of a product of variables is equal to the sum of their individual complements.

**DUALITY:-**

The implication of the duality concept is that once a theorem or statement is proved, the dual also thus stand proved. This is called the principle of duality.

$[f(A, B, C, \dots, 0, 1, +, \cdot)]_d = f(\overline{A}, \overline{B}, \overline{C}, \dots, 1, 0, \cdot, +)$

Relations between complement and dual

$f_c(A, B, C, \dots) = f(\overline{A}, \overline{B}, \overline{C}, \dots) = f_d(\overline{\overline{A}}, \overline{\overline{B}}, \overline{\overline{C}}, \dots)$

$f_d(A, B, C, \dots) = f(\overline{\overline{A}}, \overline{\overline{B}}, \overline{\overline{C}}, \dots) = f_c(\overline{\overline{\overline{A}}}, \overline{\overline{\overline{B}}}, \overline{\overline{\overline{C}}}, \dots)$

The first relation states that the complement of a function  $f(A, B, C, \dots)$  can be obtained by complementing all the variables in the dual function  $f_d(A, B, C, \dots)$ .

The second relation states that the dual can be obtained by complementing all the literals in  $f(A, B, C, \dots)$ .

**DUALS:-**

*Given expression*

*Dual*

- |  |   |
|--|---|
| 1. $\overline{0} = 1$  | $\overline{1} = 0$  |
| 2. $0 \cdot 1 = 0$   | $1 + 0 = 1$   |
| 3. $0 \cdot 0 = 0$   | $1 + 1 = 1$   |
| 4. $1 \cdot 1 = 1$   | $0 + 0 = 0$   |
| 5. $A \cdot 0 = 0$   | $A + 1 = 1$   |
| 6. $A \cdot 1 = A$   | $A + 0 = A$   |
| 7. $A \cdot \overline{A} = 0$  | $A + \overline{A} = 1$  |
| 8. $A \cdot A = A$   | $A + \overline{A} = 1$  |
| 9. $A \cdot B = B \cdot A$   | $A + B = B + A$   |
| 10. $A \cdot (B \cdot C) = (A \cdot B) \cdot C$  | $A + (B + C) = (A + B) + C$   |
| 11. $A \cdot (B + C) = AB + AC$  | $A + BC = (A + B)(A + C)$   |
| 12. $A(A + B) = A$   | $A + AB = A$  |
| 13. $\overline{A} \cdot (\overline{A} \cdot \overline{B}) = \overline{A} \cdot \overline{B}$ | $\overline{A + A + B} = \overline{A + B}$                           |
| 14. $AB = A + B$   | $A + B = \overline{\overline{A} \cdot \overline{B}}$                |
| 15. $(A + B)(\overline{A} + C)(B + C) = (A + B)(\overline{A + C})$                           | $AB + \overline{A}C + BC = \overline{AB + \overline{A}C}$           |
| 16. $A + \overline{BC} = (A + \overline{B})(A + C)$  | $A(\overline{B} + C) = \overline{A \cdot \overline{B} + A \cdot C}$ |
| 17. $(A + C)(\overline{A} + B) = AB + \overline{A}C$   | $AC + \overline{A}B = (A + B)(\overline{A} + C)$                    |
| 18. $(A + B)(C + D) = AC + AD + BC + BD$   | $(AB + CD) = (A + C)(A + D)(B + C)(B + D)$                          |
| 19. $\overline{A + B} = \overline{AB} + \overline{AB} + \overline{AB}$                       | $\overline{AB} = (A + B)(\overline{A + B})(A + B)$                  |
| 20. $\overline{AB} + \overline{A} + AB = 0$  | $\overline{A + B} \cdot \overline{A} \cdot (A + B) = 1$             |

## SUM - OF - PRODUCTS FORM:-

- This is also called disjunctive Canonical Form (DCF) or Expanded Sum of Products Form or Canonical Sum of Products Form.
- In this form, the function is the sum of a number of products terms where each product term contains all variables of the function either in complemented or uncomplemented form.
- This can also be derived from the truth table by finding the sum of all the terms that corresponds to those combinations for which 'f' assumes the value 1.

For example

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

- The product term which contains all the variables of the functions either in complemented or uncomplemented form is called a minterm.
- The minterm is denoted as  $m_0, m_1, m_2 \dots$
- An 'n' variable function can have  $2^n$  minterms.
- Another way of representing the function in canonical SOP form is the showing the sum of minterms for which the function equals to 1.

For example

$$f(A, B, C) = m_1 + m_2 + m_3 + m_5$$

or

$$f(A, B, C) = \sum m(1, 2, 3, 5)$$

where  $\sum m$  represents the sum of all the minterms whose decimal codes are given the parenthesis.

## PRODUCT- OF - SUMS FORM:-

- This form is also called as Conjunctive Canonical Form (CCF) or Expanded Product - of - Sums Form or Canonical Product Of Sums Form.
- This is by considering the combinations for which  $f = 0$
- Each term is a sum of all the variables.
- The function  $f(A, B, C) = (A + B + C \cdot \overline{C}) + (A + B + C \cdot \overline{C})$   
 $= (\overline{A} + \overline{B} + C)(\overline{A} + \overline{B} + \overline{C})(A + B + C)(A + B + \overline{C})$
- The sum term which contains each of the 'n' variables in either complemented or uncomplemented form is called a maxterm.
- Maxterm is represented as  $M_0, M_1, M_2, \dots$

Thus CCF of 'f' may be written as

$$f(A, B, C) = M_0 \cdot M_4 \cdot M_6 \cdot M_7$$

or

$$f(A, B, C) = (0, 4, 6, 7)$$

Where represented the product of all maxterms.

## CONVERSION BETWEEN CANONICAL FORM:-

The complement of a function expressed as the sum of minterms equals the sum of minterms missing from the original function.

Example:-

$$f(A, B, C) = \sum m(0, 2, 4, 6, 7)$$

This has a complement that can be expressed as

$$\overline{f(A, B, C)} = \sum m(1, 3, 5) = m_1 + m_3 + m_5$$

If we complement f by De- Morgan's theorem we obtain 'f' in a form.

$$f = \overline{(m_1 + m_3 + m_5)} = \overline{m_1} \cdot \overline{m_3} \cdot \overline{m_5}$$

$$= M_1 M_3 M_5 = \prod M(1, 3, 5)$$

**Example:-**

**Expand  $A(\bar{A} + B)(\bar{A} + B + \bar{C})$  to maxterms and minterms.**

**Solution:-**

In POS form

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

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

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

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

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

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

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

Therefore

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

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

$$= (000)(001)(010)(011)(100)(101)$$

$$= M_0 \cdot M_1 \cdot M_2 \cdot M_3 \cdot M_4 \cdot M_5$$

$$= \prod M(0, 1, 2, 3, 4, 5)$$

The maxterms  $M_6$  and  $M_7$  are missing in the POS form.

So, the SOP form will contain the minterms 6 and 7

### **KARNAUGH MAP OR K- MAP:-**

- The K- map is a chart or a graph, composed of an arrangement of adjacent cells, each representing a particular combination of variables in sum or product form.
- The K- map is systematic method of simplifying the Boolean expression.

### **TWO VARIABLE K- MAP:-**

A two variable expression can have  $2^2 = 4$  possible combinations of the input variables A and B.

**Mapping of SOP Expression:-**

- The 2 variable K-map has  $2^2 = 4$  squares. These squares are called cells.
- A '1' is placed in any square indicates that corresponding minterm is included in the output expression, and a 0 or no entry in any square indicates that the corresponding minterm does not appear in the expression for output.

|          |   |                  |            |
|----------|---|------------------|------------|
|          |   | <b>B</b>         |            |
|          |   | 0                | 1          |
| <b>A</b> | 0 | $\bar{A}\bar{B}$ | $\bar{A}B$ |
|          | 1 | $A\bar{B}$       | $AB$       |

**Example:-**

**Map expression  $f = \bar{A}B + A\bar{B}$**

**Solution:-**

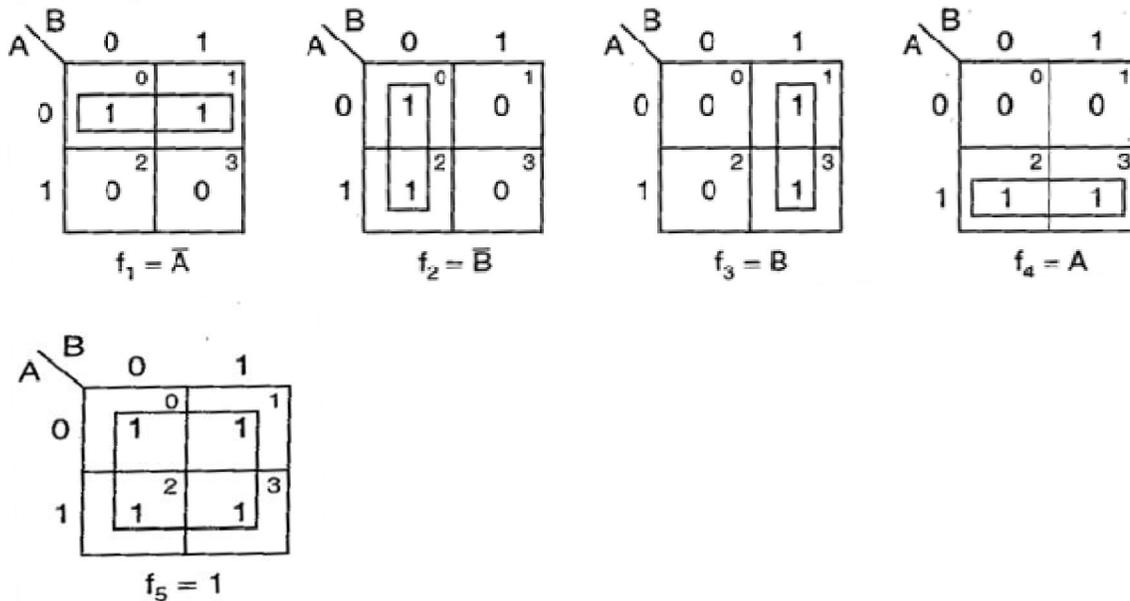
The expression minterms is

$$F = m_1 + m_2 = m(1, 2)$$

|          |   |          |   |
|----------|---|----------|---|
|          |   | <b>B</b> |   |
|          |   | 0        | 1 |
| <b>A</b> | 0 | 0        | 1 |
|          | 1 | 1        | 0 |

### Minimization of SOP Expression:-

To minimize a Boolean expression given in the SOP form by using K- map, the adjacent squares having 1s, that is minterms adjacent to each other are combined to form larger squares to eliminate some variables. The possible minterm grouping in a two variable K- map are shown below



- Two minterms, which are adjacent to each other, can be combined to form a bigger square called 2 – square or a pair. This eliminates one variable that is not common to both the minterms.
- Two 2-squares adjacent to each other can be combined to form a 4- square. A 4- square eliminates 2 variables. A 4-square is called a quad.
- Consider only those variables which remain constant throughout the square, and ignore the variables which are varying. The non-complemented variable is the variable remaining constant as 1. The complemented variable is the variable remaining constant as a 0 and the variables are written as a product term.

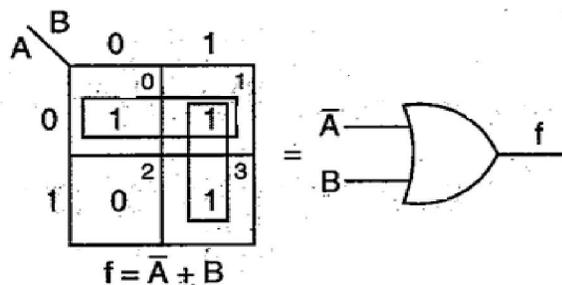
### Example:-

Reduce the expression  $f = \bar{A}B + A\bar{B} + AB$  using mapping.

### Solution:-

Expressed in terms of minterms, the given expression is

$$f = m_0 + m_1 + m_3 = \sum m (0, 1, 3)$$



$$F = \bar{A} + B$$

## Mapping of POS Expression:-

Each sum term in the standard POS expression is called a Maxterm. A function in two variables (A,B) has 4 possible maxterms,  $A + B$ ,  $A + \bar{B}$ ,  $\bar{A} + B$  and  $\bar{A} + \bar{B}$ . They are represented as  $M_0$ ,  $M_1$ ,  $M_2$  and  $M_3$  respectively.

|   |   |                            |                                  |
|---|---|----------------------------|----------------------------------|
|   |   | B                          |                                  |
|   |   | 0                          | 1                                |
| A | 0 | $A + B$ <sup>0</sup>       | $A + \bar{B}$ <sup>1</sup>       |
|   | 1 | $\bar{A} + B$ <sup>2</sup> | $\bar{A} + \bar{B}$ <sup>3</sup> |

## The maxterm of a two variable K-map

Example:-

Plot the expression  $f = (A + B)(\bar{A} + B)(\bar{A} + \bar{B})$

Solution:-

Expression in terms of maxterms is  $f = \prod M(0, 2, 3)$

|   |   |                |                |
|---|---|----------------|----------------|
|   |   | B              |                |
|   |   | 0              | 1              |
| A | 0 | 0 <sup>0</sup> | 1 <sup>1</sup> |
|   | 1 | 0 <sup>2</sup> | 0 <sup>3</sup> |

## Minimization of POS Expressions:-

In POS form the adjacent 0s are combined into large square as possible. If the squares having complemented variable then the value remain constant as a 1 and the non-complemented variable if its value remains constant as a 0 along the entire square and then their sum term is written.

The possible maxterms grouping in a two variable K-map are shown below

|   |   |   |   |
|---|---|---|---|
|   |   | B |   |
|   |   | 0 | 1 |
| A | 0 | 0 | 0 |
|   | 1 | 1 | 1 |

$f_1 = A$

|   |   |   |   |
|---|---|---|---|
|   |   | B |   |
|   |   | 0 | 1 |
| A | 0 | 1 | 0 |
|   | 1 | 1 | 0 |

$f_2 = \bar{B}$

|   |   |   |   |
|---|---|---|---|
|   |   | B |   |
|   |   | 0 | 1 |
| A | 0 | 0 | 1 |
|   | 1 | 0 | 1 |

$f_3 = B$

|   |   |   |   |
|---|---|---|---|
|   |   | B |   |
|   |   | 0 | 1 |
| A | 0 | 1 | 1 |
|   | 1 | 0 | 0 |

$f_4 = \bar{A}$

|   |   |   |   |
|---|---|---|---|
|   |   | B |   |
|   |   | 0 | 1 |
| A | 0 | 0 | 0 |
|   | 1 | 0 | 0 |

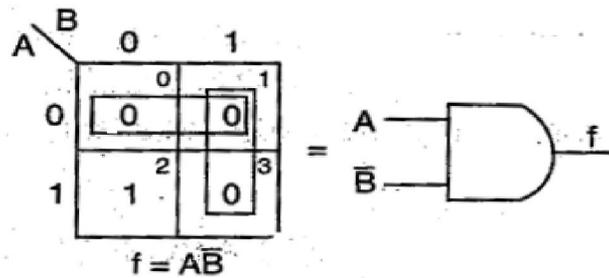
$f_5 = 0$

Example:-

Reduce the expression  $f = (A + \bar{B})(\bar{A} + \bar{B})(A + B)$  using mapping

Solution:-

The given expression in terms of maxterms is  $f = \prod M (0, 1, 3)$



### THREE VARIABLE K- MAP:-

A function in three variables (A, B, C) can be expressed in SOP and POS form having eight possible combination. A three variable K- map have 8 squares or cells and each square on the map represents a minterm or maxterm is shown in the figure below.

|   |   |                                      |                                |                                |                          |
|---|---|--------------------------------------|--------------------------------|--------------------------------|--------------------------|
|   |   | BC                                   |                                |                                |                          |
|   |   | 00                                   | 01                             | 11                             | 10                       |
| A | 0 | $\bar{A}\bar{B}\bar{C}$<br>( $m_0$ ) | $\bar{A}\bar{B}C$<br>( $m_1$ ) | $\bar{A}B\bar{C}$<br>( $m_3$ ) | $\bar{A}BC$<br>( $m_2$ ) |
|   | 1 | $A\bar{B}\bar{C}$<br>( $m_4$ )       | $A\bar{B}C$<br>( $m_5$ )       | $AB\bar{C}$<br>( $m_7$ )       | $ABC$<br>( $m_6$ )       |

(a) Minterms

|   |   |                            |                                  |  |                                  |
|---|---|----------------------------|----------------------------------|--|----------------------------------|
|   |   | BC                         |                                  |  |                                  |
|   |   | 00                         | 01                               | 11                                     | 10                               |
| A | 0 | $A+B+C$<br>( $M_0$ )       | $A+B+\bar{C}$<br>( $M_1$ )       | $A+\bar{B}+\bar{C}$<br>( $M_3$ )       | $A+\bar{B}+C$<br>( $M_2$ )       |
|   | 1 | $\bar{A}+B+C$<br>( $M_4$ ) | $\bar{A}+B+\bar{C}$<br>( $M_5$ ) | $\bar{A}+\bar{B}+\bar{C}$<br>( $M_7$ ) | $\bar{A}+\bar{B}+C$<br>( $M_6$ ) |

(b) Maxterms

Example:-

Map the expression  $f = \bar{A}\bar{B}C + \bar{A}B\bar{C} + \bar{A}BC + A\bar{B}\bar{C} + ABC$

Solution:-

So in the SOP form the expression is  $f = \sum m (1, 5, 2, 6, 7)$

|   |   |    |    |    |    |
|---|---|----|----|----|----|
|   |   | BC |    |    |    |
|   |   | 00 | 01 | 11 | 10 |
| A | 0 | 0  | 1  | 0  | 1  |
|   | 1 | 0  | 1  | 1  | 1  |

Example:-

Map the expression  $f = (A + B + C)(\bar{A} + B + C)(\bar{A} + \bar{B} + C)(A + \bar{B} + C)(\bar{A} + \bar{B} + C)$

Solution:-

So in the POS form the expression is  $f = \prod M (0, 5, 7, 3, 6)$

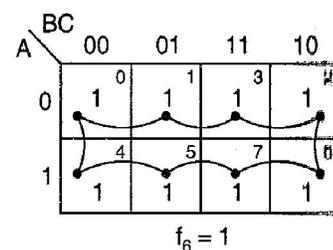
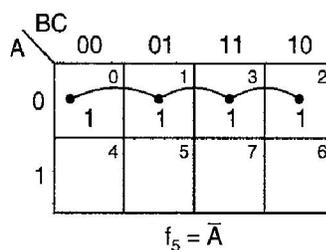
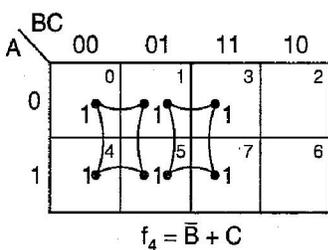
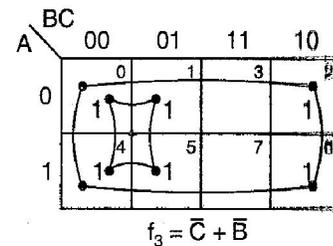
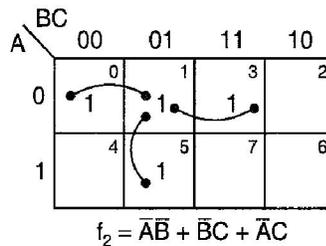
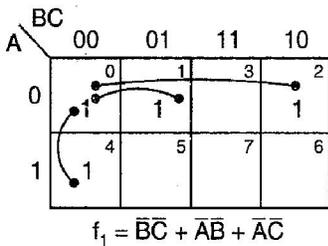
|   |   |                |                |                |                |
|---|---|----------------|----------------|----------------|----------------|
|   |   | BC             |                |                |                |
|   |   | 00             | 01             | 11             | 10             |
| A | 0 | 0 <sup>0</sup> | 1 <sup>1</sup> | 0 <sup>3</sup> | 1 <sup>2</sup> |
|   | 1 | 1 <sup>4</sup> | 0 <sup>5</sup> | 0 <sup>7</sup> | 0 <sup>6</sup> |

### Minimization of SOP and POS Expressions:-

For reducing the Boolean expressions in SOP (POS) form the following steps are given below

- Draw the K-map and place 1s (0s) corresponding to the minterms (maxterms) of the SOP (POS) expression.
- In the map 1s (0s) which are not adjacent to any other 1(0) are the isolated minterms (maxterms). They are to be read as they are because they cannot be combined even into a 2-square.
- For those 1s (0s) which are adjacent to only one other 1(0) make them pairs (2 squares).
- For quads (4- squares) and octet (8 squares) of adjacent 1s (0s) even if they contain some 1s (0s) which have already been combined. They must geometrically form a square or a rectangle.
- For any 1s (0s) that have not been combined yet then combine them into bigger squares if possible.
- Form the minimal expression by summing (multiplying) the product (sum) terms of all the groups.

Some of the possible combinations of minterms in SOP form



These possible combinations are also for POS but 1s are replaced by 0s.

### FOUR VARIABLE K-MAP:-

A four variable (A, B, C, D) expression can have  $2^4 = 16$  possible combinations of input variables. A four variable K-map has  $2^4 = 16$  squares or cells and each square on the map represents either a minterm or a maxterm as shown in the figure below. The binary number designations of the rows and columns are in the gray code. The binary numbers along the top of the map indicate the conditions of C and D along any column and binary numbers along left side indicate the conditions of A and B along any row. The numbers in the top right corners of the squares indicate the minterm or maxterm designations.

## SOP FORM

|         |  |   |   |  |
|---------|--|---|---|--|
| AB \ CD | 00   | 01  | 11  | 10   |
| 00      | $\overline{A}\overline{B}\overline{C}\overline{D}$ <sup>0</sup><br>( $m_0$ ) | $\overline{A}\overline{B}\overline{C}D$ <sup>1</sup><br>( $m_1$ ) | $\overline{A}\overline{B}C\overline{D}$ <sup>3</sup><br>( $m_3$ ) | $\overline{A}\overline{B}CD$ <sup>2</sup><br>( $m_2$ ) |
| 01      | $\overline{A}\overline{B}C\overline{D}$ <sup>4</sup><br>( $m_4$ )            | $\overline{A}\overline{B}CD$ <sup>5</sup><br>( $m_5$ )            | $\overline{A}BC\overline{D}$ <sup>7</sup><br>( $m_7$ )            | $\overline{A}BCD$ <sup>6</sup><br>( $m_6$ )            |
| 11      | $A\overline{B}\overline{C}\overline{D}$ <sup>12</sup><br>( $m_{12}$ )        | $A\overline{B}\overline{C}D$ <sup>13</sup><br>( $m_{13}$ )        | $A\overline{B}C\overline{D}$ <sup>15</sup><br>( $m_{15}$ )        | $A\overline{B}CD$ <sup>14</sup><br>( $m_{14}$ )        |
| 10      | $A\overline{B}C\overline{D}$ <sup>8</sup><br>( $m_8$ )                       | $A\overline{B}CD$ <sup>9</sup><br>( $m_9$ )                       | $ABC\overline{D}$ <sup>11</sup><br>( $m_{11}$ )                   | $ABCD$ <sup>10</sup><br>( $m_{10}$ )                   |

SOP form

## POS FORM

|         |   |  |   |  |
|---------|---|--|---|--|
| AB \ CD | 00  | 01   | 11  | 10   |
| 00      | $A+B+C+D$ <sup>0</sup><br>( $M_0$ )                           | $A+B+C+\overline{D}$ <sup>1</sup><br>( $M_1$ )                           | $A+B+\overline{C}+\overline{D}$ <sup>3</sup><br>( $M_3$ )                           | $A+B+\overline{C}+D$ <sup>2</sup><br>( $M_2$ )                           |
| 01      | $A+\overline{B}+C+D$ <sup>4</sup><br>( $M_4$ )                | $A+\overline{B}+C+\overline{D}$ <sup>5</sup><br>( $M_5$ )                | $A+\overline{B}+\overline{C}+\overline{D}$ <sup>7</sup><br>( $M_7$ )                | $A+\overline{B}+\overline{C}+D$ <sup>6</sup><br>( $M_6$ )                |
| 11      | $\overline{A}+\overline{B}+C+D$ <sup>12</sup><br>( $M_{12}$ ) | $\overline{A}+\overline{B}+C+\overline{D}$ <sup>13</sup><br>( $M_{13}$ ) | $\overline{A}+\overline{B}+\overline{C}+\overline{D}$ <sup>15</sup><br>( $M_{15}$ ) | $\overline{A}+\overline{B}+\overline{C}+D$ <sup>14</sup><br>( $M_{14}$ ) |
| 10      | $\overline{A}+B+C+D$ <sup>8</sup><br>( $M_8$ )                | $\overline{A}+B+C+\overline{D}$ <sup>9</sup><br>( $M_9$ )                | $\overline{A}+B+\overline{C}+\overline{D}$ <sup>11</sup><br>( $M_{11}$ )            | $\overline{A}+B+\overline{C}+D$ <sup>10</sup><br>( $M_{10}$ )            |

### Minimization of SOP and POS Expressions:-

For reducing the Boolean expressions in SOP (POS) form the following steps are given below

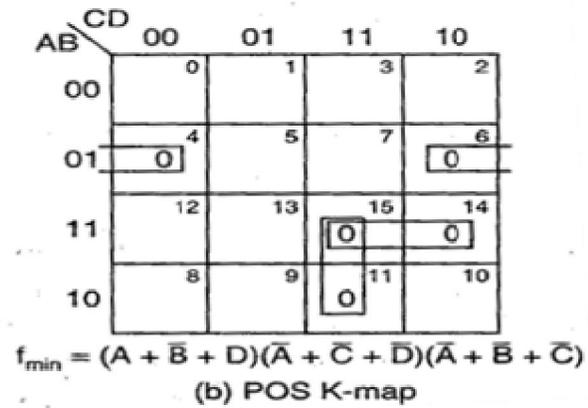
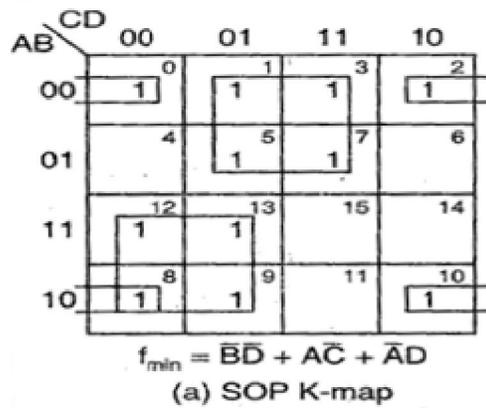
- Draw the K-map and place 1s (0s) corresponding to the minterms (maxterms) of the SOP (POS) expression.
- In the map 1s (0s) which are not adjacent to any other 1(0) are the isolated minterms (maxterms). They are to be read as they are because they cannot be combined even into a 2-square.
- For those 1s (0s) which are adjacent to only one other 1(0) make them pairs (2 squares).
- For quads (4- squares) and octet (8 squares) of adjacent 1s (0s) even if they contain some 1s (0s) which have already been combined. They must geometrically form a square or a rectangle.
- For any 1s (0s) that have not been combined yet then combine them into bigger squares if possible.
- Form the minimal expression by summing (multiplying) the product (sum) terms of all the groups.

**Example:-**

**Reduce using mapping the expression  $f = \sum m (0, 1, 2, 3, 5, 7, 8, 9, 10, 12, 13)$**

**Solution:-**

The given expression in POS form is  $f = \prod M (4, 6, 11, 14, 15)$  and in SOP form  $f = \sum m (0, 1, 2, 3, 5, 7, 8, 9, 10, 12, 13)$



The minimal SOP expression is  $f_{min} = \overline{B}D + \overline{A}C + \overline{A}D$

The minimal POS expression is  $f_{min} = (A + \overline{B} + D)(\overline{A} + \overline{C} + \overline{D})(\overline{A} + \overline{B} + \overline{C})$

### DON'T CARE COMBINATIONS:-

The combinations for which the values of the expression are not specified are called don't care combinations or optional combinations and such expression stand incompletely specified. The output is a don't care for these invalid combinations. The don't care terms are denoted by d or X. During the process of designing using SOP maps, each don't care is treated as 1 to reduce the map otherwise it is treated as 0 and left alone. During the process of designing using POS maps, each don't care is treated as 0 to reduce the map otherwise it is treated as 1 and left alone.

A standard SOP expression with don't cares can be converted into standard POS form by keeping the don't cares as they are, and the missing minterms of the SOP form are written as the maxterms of the POS form. Similarly, to convert a standard POS expression with don't cares can be converted into standard SOP form by keeping the don't cares as they are, and the missing maxterms of the POS form are written as the minterms of the SOP form.

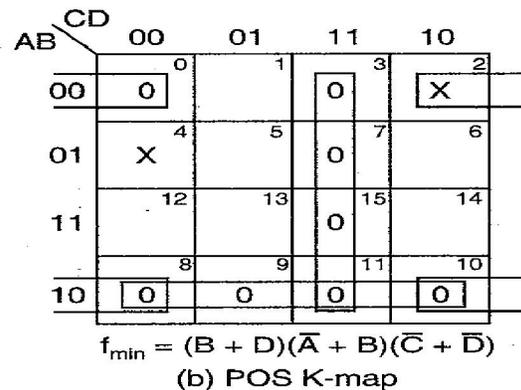
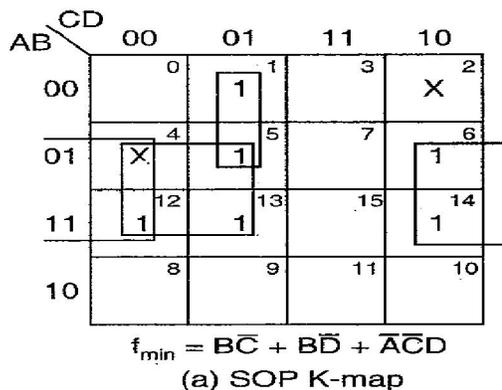
#### Example:-

Reduce the expression  $f = \sum m(1, 5, 6, 12, 13, 14) + d(2, 4)$  using K- map.

#### Solution:-

The given expression in SOP form is  $f = \sum m(1, 5, 6, 12, 13, 14) + d(2, 4)$

The given expression in POS form is  $f = \prod M(0, 3, 7, 8, 9, 10, 11, 15) + d(2, 4)$



The minimal of SOP expression is  $f_{min} = B\overline{C} + \overline{B}D + \overline{A}\overline{C}D$

The minimal of POS expression is  $f_{min} = (B + D)(\overline{A} + \overline{B})(\overline{C} + \overline{D})$

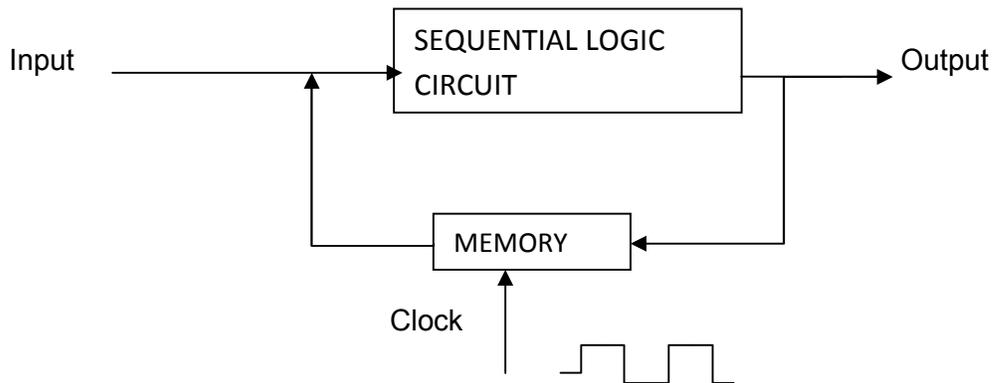
# **SEQUENTIAL LOGIC CIRCUIT**

## SEQUENTIAL CIRCUIT:-

- It is a circuit whose output depends upon the present input, previous output and the sequence in which the inputs are applied.

## HOW THE SEQUENTIAL CIRCUIT IS DIFFERENT FROM COMBINATIONAL CIRCUIT? :-

- In combinational circuit output depends upon present input at any instant of time and do not use memory. Hence previous input does not have any effect on the circuit. But sequential circuit has memory and depends upon present input and previous output.
- Sequential circuits are slower than combinational circuits and these sequential circuits are harder to design.



[Block diagram of Sequential Logic Circuit]

- The data stored by the memory element at any given instant of time is called the present state of sequential circuit.

## TYPES:-

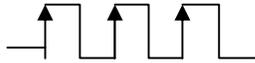
Sequential logic circuits (SLC) are classified as

- (i) Synchronous SLC
  - (ii) Asynchronous SLC
- The SLC that are controlled by clock are called synchronous SLC and those which are not controlled by a clock are asynchronous SLC.
  - Clock:- A recurring pulse is called a clock.

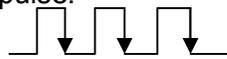
## FLIP-FLOP AND LATCH:-

- A flip-flop or latch is a circuit that has two stable states and can be used to store information.
- A flip-flop is a binary storage device capable of storing one bit of information. In a stable state, the output of a flip-flop is either 0 or 1.
- Latch is a non-clocked flip-flop and it is the building block for the flip-flop.
- A storage element in digital circuit can maintain a binary state indefinitely until directed by an input signal to switch state.
- Storage element that operate with signal level are called latches and those operate with clock transition are called as flip-flops.

- The circuit can be made to change state by signals applied to one or more control inputs and will have one or two outputs.
- A flip-flop is called so because its output either flips or flops meaning to switch back and forth.
- A flip-flop is also called a bi-stable multi-vibrator as it has two stable states. The input signals which command the flip-flop to change state are called excitations.
- Flip-flops are storage devices and can store 1 or 0.
- Flip-flops using the clock signal are called clocked flip-flops. Control signals are effective only if they are applied in synchronization with the clock signal.
- Clock-signals may be positive-edge triggered or negative-edge triggered.
- Positive-edge triggered flip-flops are those in which state transitions take place only at positive-going edge of the clock pulse.



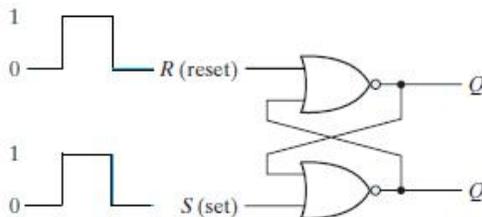
- Negative-edge triggered flip-flops are those in which state transition take place only at negative-going edge of the clock pulse.



- Some common type of flip-flops include
  - a) SR (set-reset) F-F
  - b) D (data or delay) F-F
  - c) T (toggle) F-F and
  - d) JK F-F

### SR latch:-

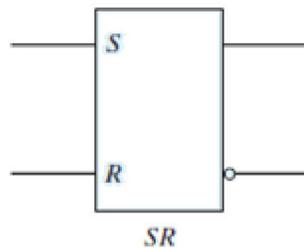
- The SR latch is a circuit with two cross-coupled NOR gates or two cross-coupled NAND gates.
- It has two outputs labeled Q and Q'. Two inputs are there labeled S for set and R for reset.
- The latch has two useful states. When Q=0 and Q'=1 the condition is called reset state and when Q=1 and Q'=0 the condition is called set state.
- Normally Q and Q' are complement of each other.
- The figure represents a SR latch with two cross-coupled NOR gates. The circuit has NOR gates and as we know if any one of the input for a NOR gate is HIGH then its output will be LOW and if both the inputs are LOW then only the output will be HIGH.



- Under normal conditions, both inputs of the latch remain at 0 unless the state has to be changed. The application of a momentary 1 to the S input causes the latch to go to the set state. The S input must go back to 0 before any other changes take place, in order to avoid the occurrence of an undefined next state that results from the forbidden input condition.
- The first condition (S = 1, R = 0) is the action that must be taken by input S to bring the circuit to the set state. Removing the active input from S leaves the circuit in the same state. After both inputs return to 0, it is then possible to shift to the reset state by momentarily applying a 1 to the R input. The 1 can then be removed from R, whereupon the circuit remains in the reset state. When both inputs S and R are equal to 0, the latch can be in either the set or the reset state, depending on which input was most recently a 1.

- If a 1 is applied to both the S and R inputs of the latch, both outputs go to 0. This action produces an undefined next state, because the state that results from the input transitions depends on the order in which they return to 0. It also violates the requirement that outputs be the complement of each other. In normal operation, this condition is avoided by making sure that 1's are not applied to both inputs simultaneously.
- Truth table for SR latch designed with NOR gates is shown below.

| Input |   | Output |    |                   |                    | Comment          |
|-------|---|--------|----|-------------------|--------------------|------------------|
| S     | R | Q      | Q' | Q <sub>Next</sub> | Q' <sub>Next</sub> |                  |
| 0     | 0 | 0      | 1  | 0                 | 1                  | No change        |
| 0     | 0 | 1      | 0  | 1                 | 0                  |                  |
| 0     | 1 | 0      | 1  | 0                 | 1                  | Reset            |
| 0     | 1 | 1      | 0  | 0                 | 1                  |                  |
| 1     | 0 | 0      | 1  | 1                 | 0                  | Set              |
| 1     | 0 | 1      | 0  | 1                 | 0                  |                  |
| 1     | 1 | 0      | 1  | X                 | X                  | Prohibited state |
| 1     | 1 | 1      | 0  | X                 | X                  |                  |



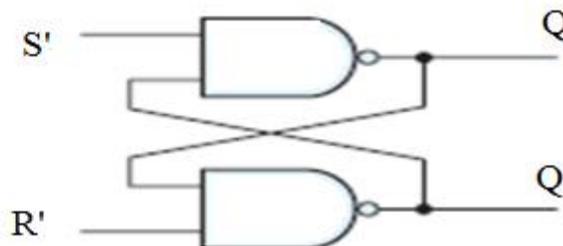
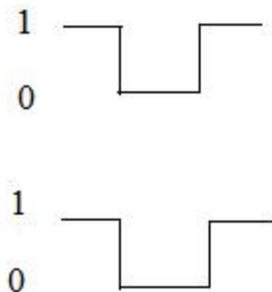
Symbol for SR NOR Latch

**Racing Condition:-**

In case of a SR latch when S=R=1 input is given both the output will try to become 0. This is called Racing condition.

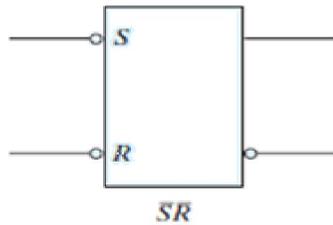
**SR latch using NAND gate:-**

- The below figure represents a SR latch with two cross-coupled NAND gates. The circuit has NAND gates and as we know if any one of the input for a NAND gate is LOW then its output will be HIGH and if both the inputs are HIGH then only the output will be LOW.
- It operates with both inputs normally at 1, unless the state of the latch has to be changed. The application of 0 to the S input causes output Q to go to 1, putting the latch in the set state. When the S input goes back to 1, the circuit remains in the set state. After both inputs go back to 1, we are allowed to change the state of the latch by placing a 0 in the R input. This action causes the circuit to go to the reset state and stay there even after both inputs return to 1.



- The condition that is forbidden for the NAND latch is both inputs being equal to 0 at the same time, an input combination that should be avoided.

In comparing the NAND with the NOR latch, note that the input signals for the NAND require the complement of those values used for the NOR latch. Because the NAND latch requires a 0 signal to change its state, it is sometimes referred to as an S'R' latch. The primes (or, sometimes, bars over the letters) designate the fact that the inputs must be in their complement form to activate the circuit.

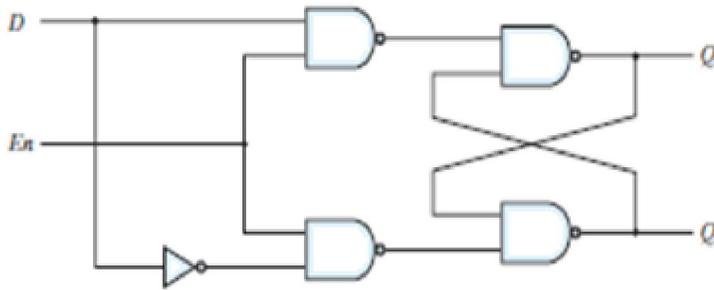


The above represents the symbol for inverted SR latch or SR latch using NAND gate.  
 Truth table for SR latch using NAND gate or Inverted SR latch

| S | R | Q <sub>next</sub> | Q' <sub>next</sub> |
|---|---|-------------------|--------------------|
| 0 | 0 | Race              | Race               |
| 0 | 1 | 0                 | 1 (Reset)          |
| 1 | 0 | 1                 | 0 (Set)            |
| 1 | 1 | Q (No change)     | Q' (No change)     |

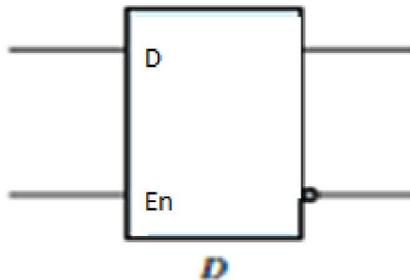
**D LATCH:-**

- One way to eliminate the undesirable condition of the indeterminate state in the SR latch is to ensure that inputs S and R are never equal to 1 at the same time.



(a) Logic diagram

- This is done in the D latch. This latch has only two inputs: D (data) and En (enable).
- The D input goes directly to the S input, and its complement is applied to the R input.



(Symbol for D-Latch)

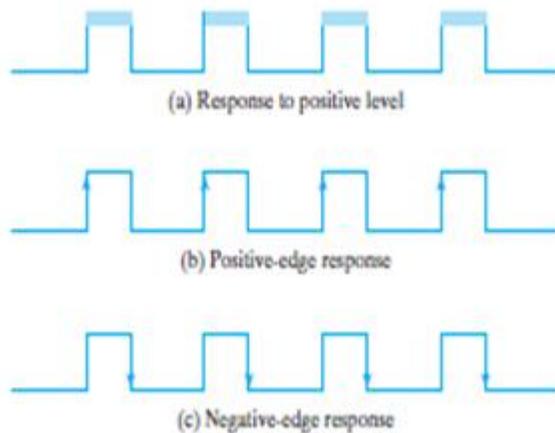
- As long as the enable input is at 0, the cross-coupled SR latch has both inputs at the 1 level and the circuit can't change state regardless of the value of D.
- The below represents the truth table for the D-latch.

| En | D | Next State of Q |
|----|---|-----------------|
| 0  | X | No change       |
| 1  | 0 | Q=0;Reset State |
| 1  | 1 | Q=1;Set State   |

- The D input is sampled when  $En = 1$ . If  $D = 1$ , the Q output goes to 1, placing the circuit in the set state. If  $D = 0$ , output Q goes to 0, placing the circuit in the reset state. This situation provides a path from input D to the output, and for this reason, the circuit is often called a TRANSPARENT latch.

### TRIGGERING METHODS:-

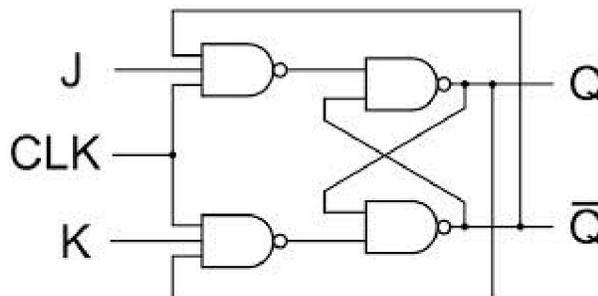
- The state of a latch or flip-flop is switched by a change in the control input. This momentary change is called a trigger, and the transition it causes is said to trigger the flip-flop.
- Flip-flop circuits are constructed in such a way as to make them operate properly when they are part of a sequential circuit that employs a common clock.
- The problem with the latch is that it responds to a change in the level of a clock pulse. For proper operation of a flip-flop it should be triggered only during a signal transition.
- This can be accomplished by eliminating the feedback path that is inherent in the operation of the sequential circuit using latches. A clock pulse goes through two transitions: from 0 to 1 and the return from 1 to 0.
- A ways that a latch can be modified to form a flip-flop is to produce a flip-flop that triggers only during a signal transition (from 0 to 1 or from 1 to 0) of the synchronizing signal (clock) and is disabled during the rest of the clock pulse.



### JK FLIP-FLOP:-

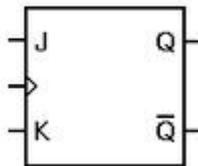
- The JK flip-flop can be constructed by using basic SR latch and a clock. In this case the outputs Q and Q' are returned back and connected to the inputs of NAND gates.
- This simple JK flip Flop is the most widely used of all the flip-flop designs and is considered to be a universal flip-flop circuit.
- The sequential operation of the JK flip flop is exactly the same as for the previous SR flip-flop with the same "Set" and "Reset" inputs.
- The difference this time is that the "JK flip flop" has no invalid or forbidden input states of the SR Latch even when S and R are both at logic "1".

(The below diagram shows the circuit diagram of a JK flip-flop)



- The JK flip flop is basically a gated SR Flip-flop with the addition of a clock input circuitry that prevents the illegal or invalid output condition that can occur when both inputs S and R are equal to logic level "1".
- Due to this additional clocked input, a JK flip-flop has four possible input combinations, "logic 1", "logic 0", "no change" and "toggle".

- The symbol for a JK flip flop is similar to that of an SR bistable latch except the clock input.



(The above diagram shows the symbol of a JK flip-flop.)

- Both the S and the R inputs of the SR bi-stable have now been replaced by two inputs called the J and K inputs, respectively after its inventor Jack and Kilby. Then this equates to:  $J = S$  and  $K = R$ .
- The two 2-input NAND gates of the gated SR bi-stable have now been replaced by two 3-input NAND gates with the third input of each gate connected to the outputs at Q and Q'.
- This cross coupling of the SR flip-flop allows the previously invalid condition of  $S = "1"$  and  $R = "1"$  state to be used to produce a "toggle action" as the two inputs are now interlocked.
- If the circuit is now "SET" the J input is inhibited by the "0" status of Q' through the lower NAND gate. If the circuit is "RESET" the K input is inhibited by the "0" status of Q through the upper NAND gate. As Q and Q' are always different we can use them to control the input.

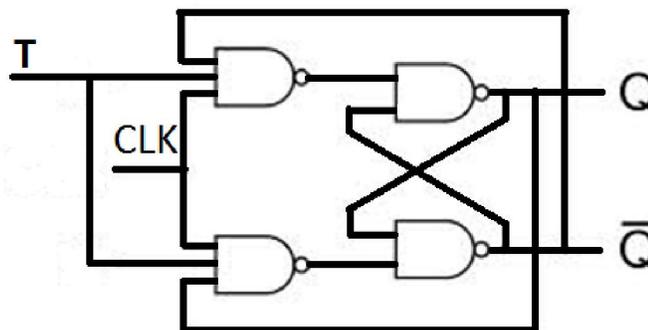
(Truth table for JK flip-flop)

| Input |   | Output |                   | Comment   |
|-------|---|--------|-------------------|-----------|
| J     | K | Q      | Q <sub>next</sub> |           |
| 0     | 0 | 0      | 0                 | No change |
| 0     | 0 | 1      | 1                 |           |
| 0     | 1 | 0      | 0                 | Reset     |
| 0     | 1 | 1      | 0                 |           |
| 1     | 0 | 0      | 1                 | Set       |
| 1     | 0 | 1      | 1                 |           |
| 1     | 1 | 0      | 1                 | Toggle    |
| 1     | 1 | 1      | 0                 |           |

- When both inputs J and K are equal to logic "1", the JK flip flop toggles.

### T FLIP-FLOP:-

- Toggle flip-flop or commonly known as T flip-flop.
- This flip-flop has the similar operation as that of the JK flip-flop with both the inputs J and K are shorted i.e. both are given the common input.



- Hence its truth table is same as that of JK flip-flop when  $J=K= 0$  and  $J=K=1$ . So its truth table is as follows.

| T | Q | Q <sub>next</sub> | Comment   |
|---|---|-------------------|-----------|
| 0 | 0 | 0                 | No change |
|   | 1 | 1                 |           |
| 1 | 0 | 1                 | Toggles   |
|   | 1 | 0                 |           |

### CHARACTERISTIC TABLE:-

- A characteristic table defines the logical properties of a flip-flop by describing its operation in tabular form.
- The next state is defined as a function of the inputs and the present state.
- Q (t) refers to the present state and Q (t + 1) is the next.
- Thus, Q (t) denotes the state of the flip-flop immediately before the clock edge, and Q(t + 1) denotes the state that results from the clock transition.
- The characteristic table for the JK flip-flop shows that the next state is equal to the present state when inputs J and K are both equal to 0. This condition can be expressed as  $Q(t + 1) = Q(t)$ , indicating that the clock produces no change of state.

#### Characteristic Table Of JK Flip-Flop

| J | K | Q(t+1)           |
|---|---|------------------|
| 0 | 0 | Q(t) No change   |
| 0 | 1 | 0 Reset          |
| 1 | 0 | 1 Set            |
| 1 | 1 | Q'(t) Complement |

- When K = 1 and J = 0, the clock resets the flip-flop and  $Q(t + 1) = 0$ . With J = 1 and K = 0, the flip-flop sets and  $Q(t + 1) = 1$ . When both J and K are equal to 1, the next state changes to the complement of the present state, a transition that can be expressed as  $Q(t + 1) = Q'(t)$ .
- The characteristic equation for JK flip-flop is represented as  

$$Q(t+1) = JQ' + K'Q$$

#### Characteristic Table of D Flip-Flop

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

- The next state of a D flip-flop is dependent only on the D input and is independent of the present state.
- This can be expressed as  $Q(t + 1) = D$ . It means that the next-state value is equal to the value of D. Note that the D flip-flop does not have a “no-change” condition and its characteristic equation is written as  $Q(t+1)=D$ .

#### Characteristic Table of T Flip-Flop

| T | Q(t+1)           |
|---|------------------|
| 0 | Q(t) No change   |
| 1 | Q'(t) Complement |

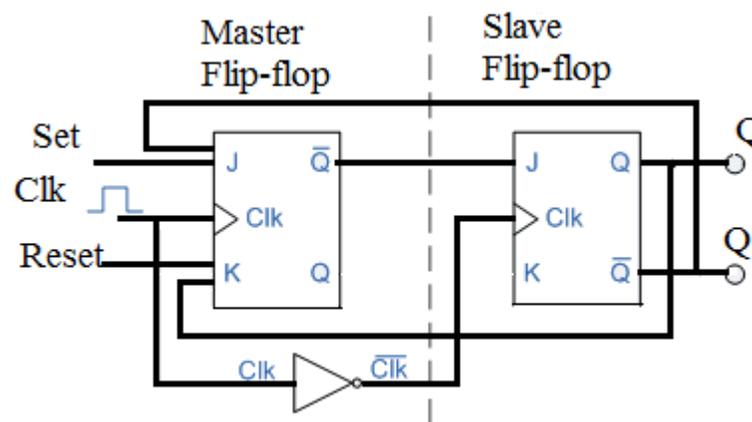
- The characteristic table of T flip-flop has only two conditions: When  $T = 0$ , the clock edge does not change the state; when  $T = 1$ , the clock edge complements the state of the flip-flop and the characteristic equation is

$$Q(t+1) = T \oplus Q = T'Q + TQ'$$

### MASTER-SLAVE JK FLIP-FLOP:-

- The Master-Slave Flip-Flop is basically two gated SR flip-flops connected together in a series configuration with the slave having an inverted clock pulse.
- The outputs from Q and Q' from the "Slave" flip-flop are fed back to the inputs of the "Master" with the outputs of the "Master" flip flop being connected to the two inputs of the "Slave" flip flop.
- This feedback configuration from the slave's output to the master's input gives the characteristic toggle of the JK flip flop as shown below.

The Master-Slave JK Flip Flop



- The input signals J and K are connected to the gated "master" SR flip flop which "locks" the input condition while the clock (Clk) input is "HIGH" at logic level "1".
- As the clock input of the "slave" flip flop is the inverse (complement) of the "master" clock input, the "slave" SR flip flop does not toggle.
- The outputs from the "master" flip flop are only "seen" by the gated "slave" flip flop when the clock input goes "LOW" to logic level "0".
- When the clock is "LOW", the outputs from the "master" flip flop are latched and any additional changes to its inputs are ignored.
- The gated "slave" flip flop now responds to the state of its inputs passed over by the "master" section.
- Then on the "Low-to-High" transition of the clock pulse the inputs of the "master" flip flop are fed through to the gated inputs of the "slave" flip flop and on the "High-to-Low" transition the same inputs are reflected on the output of the "slave" making this type of flip flop edge or pulse-triggered.
- Then, the circuit accepts input data when the clock signal is "HIGH", and passes the data to the output on the falling-edge of the clock signal.
- In other words, the Master-Slave JK Flip flop is a "Synchronous" device as it only passes data with the timing of the clock signal.

### FLIP-FLOP CONVERSIONS:-

#### SR Flip Flop to JK Flip Flop

For this J and K will be given as external inputs to S and R. As shown in the logic diagram below, S and R will be the outputs of the combinational circuit.

The truth tables for the flip flop conversion are given below. The present state is represented by  $Q_p$  and  $Q_{p+1}$  is the next state to be obtained when the J and K inputs are applied.

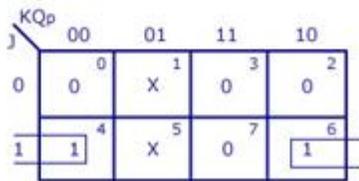
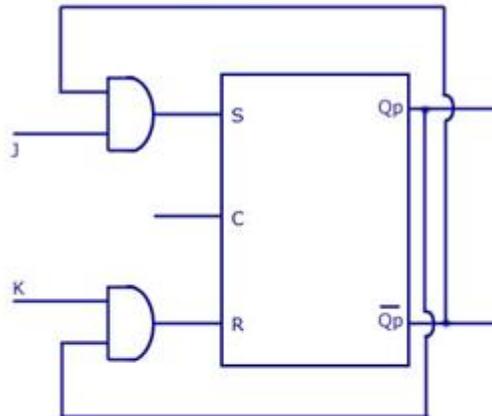
For two inputs J and K, there will be eight possible combinations. For each combination of J, K and  $Q_p$ , the corresponding  $Q_{p+1}$  states are found.  $Q_{p+1}$  simply suggests the future values to be obtained by the JK flip flop after the value of  $Q_p$ . The table is then completed by writing the values of S and R required to get each  $Q_{p+1}$  from the corresponding  $Q_p$ . That is, the values of S and R that are required to change the state of the flip flop from  $Q_p$  to  $Q_{p+1}$  are written.

### S-R Flip Flop to J-K Flip Flop

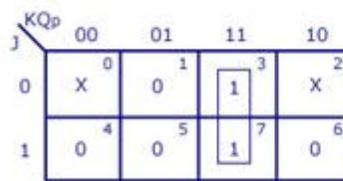
Conversion Table

| J-K Inputs |   | Outputs |           | S-R Inputs |   |
|------------|---|---------|-----------|------------|---|
| J          | K | $Q_p$   | $Q_{p+1}$ | S          | R |
| 0          | 0 | 0       | 0         | 0          | X |
| 0          | 0 | 1       | 1         | X          | 0 |
| 0          | 1 | 0       | 0         | 0          | X |
| 0          | 1 | 1       | 0         | 0          | 1 |
| 1          | 0 | 0       | 1         | 1          | 0 |
| 1          | 0 | 1       | 1         | X          | 0 |
| 1          | 1 | 0       | 1         | 1          | 0 |
| 1          | 1 | 1       | 0         | 0          | 1 |

Logic Diagram



$$S = \bar{J}Q_p$$



$$R = KQ_p$$

K-Map

### JK Flip Flop to SR Flip Flop

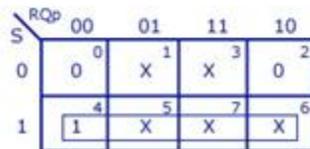
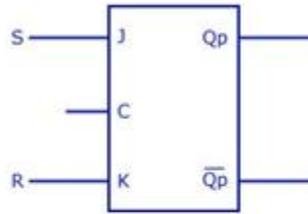
- This will be the reverse process of the above explained conversion. S and R will be the external inputs to J and K. J and K will be the outputs of the combinational circuit. Thus, the values of J and K have to be obtained in terms of S, R and  $Q_p$ .
- A conversion table is to be written using S, R,  $Q_p$ ,  $Q_{p+1}$ , J and K.
- For two inputs, S and R, eight combinations are made. For each combination, the corresponding  $Q_{p+1}$  outputs are found out.
- The outputs for the combinations of  $S=1$  and  $R=1$  are not permitted for an SR flip flop. Thus the outputs are considered invalid and the J and K values are taken as "don't cares".

### J-K Flip Flop to S-R Flip Flop

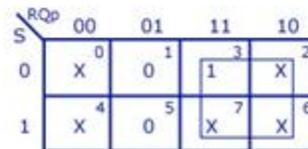
Conversion Table

| S-R Inputs |   | Outputs |      | J-K Inputs |   |
|------------|---|---------|------|------------|---|
| S          | R | Qp      | Qp+1 | J          | K |
| 0          | 0 | 0       | 0    | 0          | X |
| 0          | 0 | 1       | 1    | X          | 0 |
| 0          | 1 | 0       | 0    | 0          | X |
| 0          | 1 | 1       | 0    | X          | 1 |
| 1          | 0 | 0       | 1    | 1          | X |
| 1          | 0 | 1       | 1    | X          | 0 |
| 1          | 1 | Invalid |      | Dont care  |   |
| 1          | 1 | Invalid |      | Dont care  |   |

Logic Diagram



J=S



K-maps

K=R

### SR Flip Flop to D Flip Flop

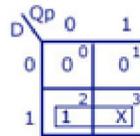
- S and R are the actual inputs of the flip flop and D is the external input of the flip flop.
- The four combinations, the logic diagram, conversion table, and the K-map for S and R in terms of D and Qp are shown below.

### S-R Flip Flop to D Flip Flop

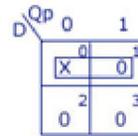
Conversion Table

| D Input | Outputs |      | S-R Inputs |   |
|---------|---------|------|------------|---|
|         | Qp      | Qp+1 | S          | R |
| 0       | 0       | 0    | 0          | X |
| 0       | 1       | 0    | 0          | 1 |
| 1       | 0       | 1    | 1          | 0 |
| 1       | 1       | 1    | X          | 0 |

K-maps

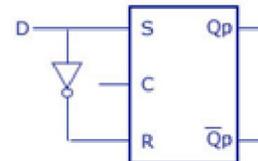


S = D



R = D-bar

Logic Diagram



### D Flip Flop to SR Flip Flop

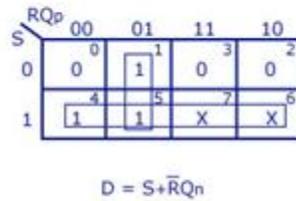
- D is the actual input of the flip flop and S and R are the external inputs. Eight possible combinations are achieved from the external inputs S, R and Qp.
- But, since the combination of S=1 and R=1 are invalid, the values of Qp+1 and D are considered as "don't cares".
- The logic diagram showing the conversion from D to SR, and the K-map for D in terms of S, R and Qp are shown below.

### D Flip Flop to S-R Flip Flop

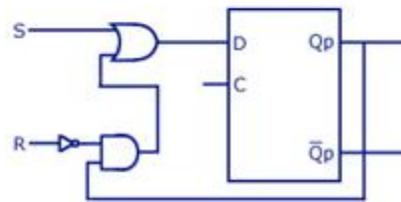
Conversion Table

| S-R Inputs |   | Outputs        |                  | D Input    |
|------------|---|----------------|------------------|------------|
| S          | R | Q <sub>p</sub> | Q <sub>p+1</sub> |            |
| 0          | 0 | 0              | 0                | 0          |
| 0          | 0 | 1              | 1                | 1          |
| 0          | 1 | 0              | 0                | 0          |
| 0          | 1 | 1              | 0                | 0          |
| 1          | 0 | 0              | 1                | 1          |
| 1          | 0 | 1              | 1                | 1          |
| 1          | 1 | Invalid        |                  | Don't care |
| 1          | 1 | Invalid        |                  | Don't care |

K-map



Logic Diagram



### JK Flip Flop to T Flip Flop:-

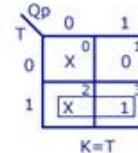
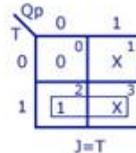
- J and K are the actual inputs of the flip flop and T is taken as the external input for conversion
- Four combinations are produced with T and Q<sub>p</sub>. J and K are expressed in terms of T and Q<sub>p</sub>.
  - The conversion table, K-maps, and the logic diagram are given below.

J-K Flip Flop to T Flip Flop

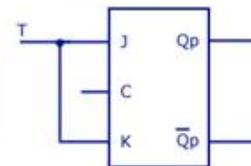
Conversion Table

| T Input | Outputs        |                  | J-K Inputs |   |
|---------|----------------|------------------|------------|---|
|         | Q <sub>p</sub> | Q <sub>p+1</sub> | J          | K |
| 0       | 0              | 0                | 0          | X |
| 0       | 1              | 1                | X          | 0 |
| 1       | 0              | 1                | 1          | X |
| 1       | 1              | 0                | X          | 1 |

K-maps



Logic Diagram



### D Flip Flop to JK Flip Flop:-

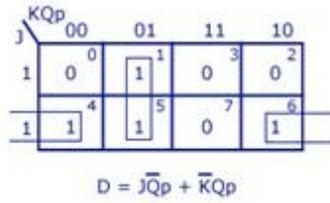
- In this conversion, D is the actual input to the flip flop and J and K are the external inputs.
- J, K and Q<sub>p</sub> make eight possible combinations, as shown in the conversion table below. D is expressed in terms of J, K and Q<sub>p</sub>.
- The conversion table, the K-map for D in terms of J, K and Q<sub>p</sub> and the logic diagram showing the conversion from D to JK are given in the figure below.

### D Flip Flop to J-K Flip Flop

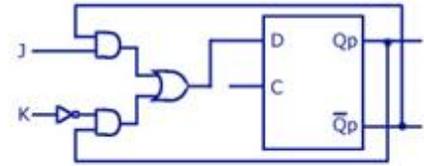
Conversion Table

| J-K Input |   | Outputs        |                  | D Input |
|-----------|---|----------------|------------------|---------|
| J         | K | Q <sub>p</sub> | Q <sub>p+1</sub> |         |
| 0         | 0 | 0              | 0                | 0       |
| 0         | 0 | 1              | 1                | 1       |
| 0         | 1 | 0              | 0                | 0       |
| 0         | 1 | 1              | 0                | 0       |
| 1         | 0 | 0              | 1                | 1       |
| 1         | 0 | 1              | 1                | 1       |
| 1         | 1 | 0              | 1                | 1       |
| 1         | 1 | 1              | 0                | 0       |

K-map



Logic Diagram



### JK Flip Flop to D Flip Flop:-

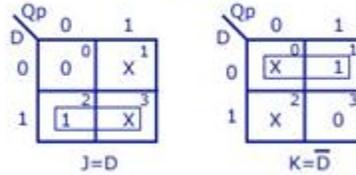
- D is the external input and J and K are the actual inputs of the flip flop. D and Q<sub>p</sub> make four combinations. J and K are expressed in terms of D and Q<sub>p</sub>.
- The four combination conversion table, the K-maps for J and K in terms of D and Q<sub>p</sub>.

### J-K Flip Flop to D Flip Flop

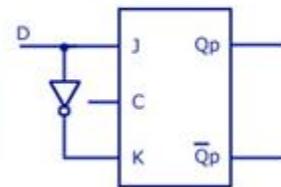
Conversion Table

| D Input | Outputs        |                  | J-K Inputs |   |
|---------|----------------|------------------|------------|---|
|         | Q <sub>p</sub> | Q <sub>p+1</sub> | J          | K |
| 0       | 0              | 0                | 0          | X |
| 0       | 1              | 0                | X          | 1 |
| 1       | 0              | 1                | 1          | X |
| 1       | 1              | 0                | X          | 0 |

K-maps



Logic Diagram



# COMBINATIONAL LOGIC CIRCUIT

- A combinational circuit consists of logic gates whose outputs at any time are determined from only the present combination of inputs.
- A combinational circuit performs an operation that can be specified logically by a set of Boolean functions.
- It consists of an interconnection of logic gates. Combinational logic gates react to the values of the signals at their inputs and produce the value of the output signal, transforming binary information from the given input data to a required output data.
- A block diagram of a combinational circuit is shown in the below figure.
- The n input binary variables come from an external source; the m output variables are produced by the internal combinational logic circuit and go to an external destination.
- Each input and output variable exists physically as an analog signal whose values are interpreted to be a binary signal that represents logic 1 and logic 0.



## BINARY ADDER-SUBTRACTOR:-

- Digital computers perform a variety of information-processing tasks. Among the functions encountered are the various arithmetic operations.
- The most basic arithmetic operation is the addition of two binary digits. This simple addition consists of four possible elementary operations:  $0 + 0 = 0$ ,  $0 + 1 = 1$ ,  $1 + 0 = 1$ , and  $1 + 1 = 10$ .
- The first three operations produce a sum of one digit, but when both augend and addend bits are equal to 1; the binary sum consists of two digits. The higher significant bit of this result is called a carry.
- When the augend and addend numbers contain more significant digits, the carry obtained from the addition of two bits is added to the next higher order pair of significant bits.
- A combinational circuit that performs the addition of two bits is called a half adder.
- One that performs the addition of three bits (two significant bits and a previous carry) is a full adder. The names of the circuits stem from the fact that two half adders can be employed to implement a full adder.

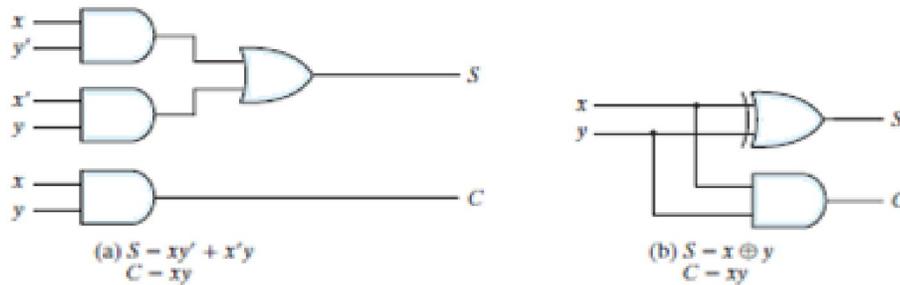
## HALF ADDER:-

- This circuit needs two binary inputs and two binary outputs.
- The input variables designate the augend and addend bits; the output variables produce the sum and carry. Symbols x and y are assigned to the two inputs and S (for sum) and C (for carry) to the outputs.
- The truth table for the half adder is listed in the below table.
- The C output is 1 only when both inputs are 1. The S output represents the least significant bit of the sum.
- The simplified Boolean functions for the two outputs can be obtained directly from the truth table.

| x | y | D | B |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

Truth Table

- The simplified sum-of-products expressions are
$$S = x'y + xy'$$
$$C = xy$$
- The logic diagram of the half adder implemented in sum of products is shown in the below figure. It can be also implemented with an exclusive-OR and an AND gate.



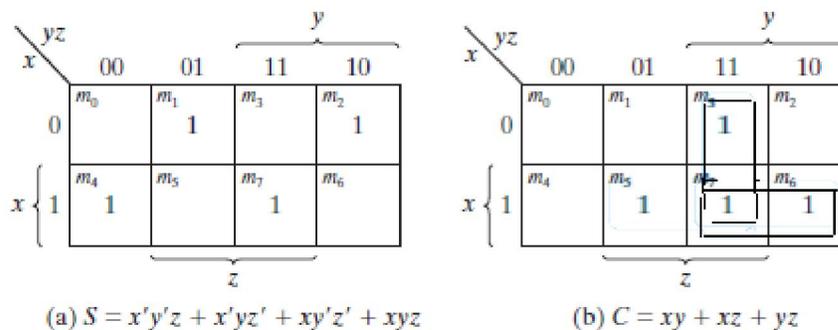
### FULL ADDER:-

- A full adder is a combinational circuit that forms the arithmetic sum of three bits.
- It consists of three inputs and two outputs. Two of the input variables, denoted by  $x$  and  $y$ , represent the two significant bits to be added. The third input,  $z$ , represents the carry from the previous lower significant position.

| $x$ | $y$ | $z$ | $C$ | $S$ |
|-----|-----|-----|-----|-----|
| 0   | 0   | 0   | 0   | 0   |
| 0   | 0   | 1   | 0   | 1   |
| 0   | 1   | 0   | 0   | 1   |
| 0   | 1   | 1   | 1   | 0   |
| 1   | 0   | 0   | 0   | 1   |
| 1   | 0   | 1   | 1   | 0   |
| 1   | 1   | 0   | 1   | 0   |
| 1   | 1   | 1   | 1   | 1   |

Truth Table

- Two outputs are necessary because the arithmetic sum of three binary digits ranges in value from 0 to 3, and binary representation of 2 or 3 needs two bits. The two outputs are designated by the symbols  $S$  for sum and  $C$  for carry.



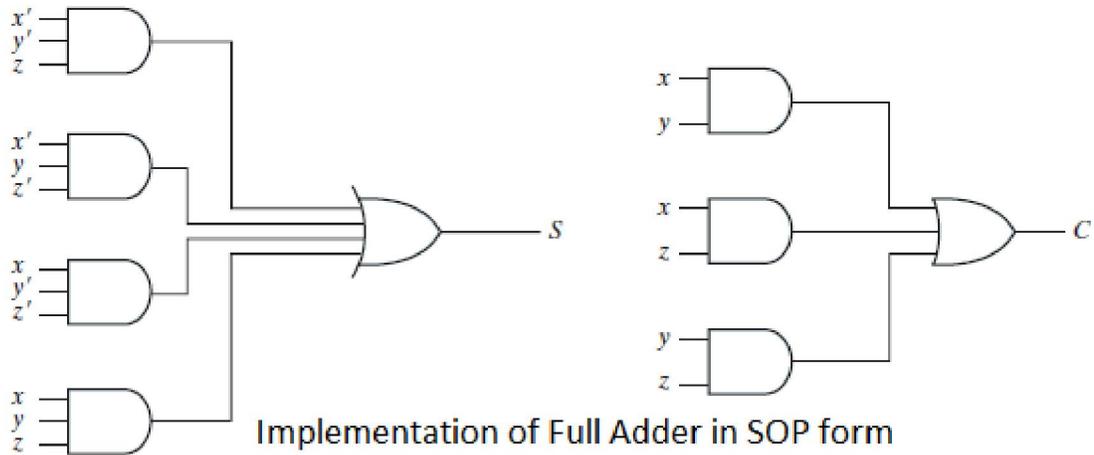
K-Map for full adder

- The binary variable  $S$  gives the value of the least significant bit of the sum. The binary variable  $C$  gives the output carry formed by adding the input carry and the bits of the words.
- The eight rows under the input variables designate all possible combinations of the three variables. The output variables are determined from the arithmetic sum of the input bits. When all input bits are 0, the output is 0.
- The  $S$  output is equal to 1 when only one input is equal to 1 or when all three inputs are equal to 1. The  $C$  output has a carry of 1 if two or three inputs are equal to 1.
- The simplified expressions are

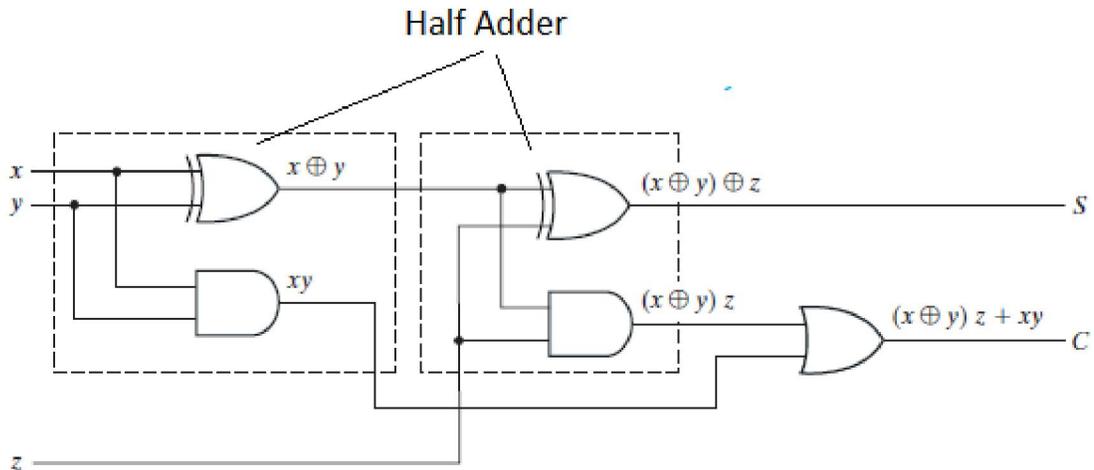
$$S = x'y'z + x'yz' + xy'z' + xyz$$

$$C = xy + xz + yz$$

- The logic diagram for the full adder implemented in sum-of-products form is shown in figure.



- It can also be implemented with two half adders and one OR gate as shown in the figure.



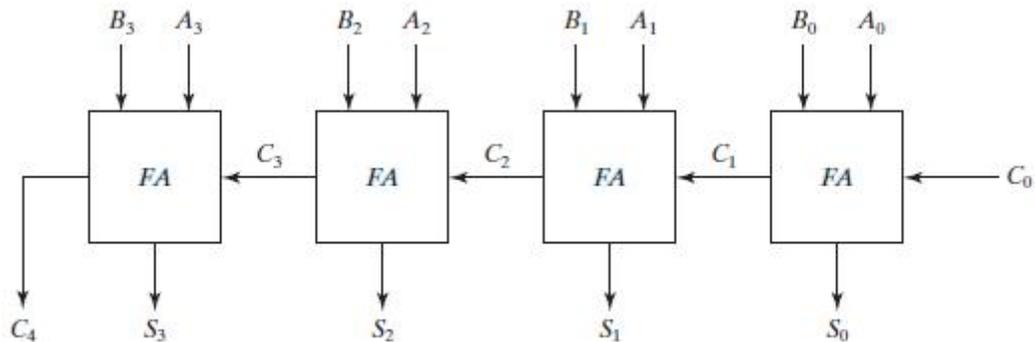
- A full adder is a combinational circuit that forms the arithmetic sum of three bits.

#### BINARY ADDER:-

- A binary adder is a digital circuit that produces the arithmetic sum of two binary numbers.
- It can be constructed with full adders connected in cascade, with the output carry from each full adder connected to the input carry of the next full adder in the chain.
- Addition of n-bit numbers requires a chain of n full adders or a chain of one-half adder and n-1 full adders. In the former case, the input carry to the least significant position is fixed at 0.
- The interconnection of four full-adder (FA) circuits to provide a four-bit binary ripple carry adder is shown in the figure.
- The augend bits of A and the addend bits of B are designated by subscript numbers from right to left, with subscript 0 denoting the least significant bit.
- The carries are connected in a chain through the full adders. The input carry to the adder is  $C_0$ , and it ripples through the full adders to the output carry  $C_4$ . The S outputs generate the required sum bits.
- An n-bit adder requires n full adders, with each output carry connected to the input carry of the next higher order full adder.
- Consider the two binary numbers  $A = 1011$  and  $B = 0011$ . Their sum  $S = 1110$  is formed with the four-bit adder as follows:

| Subscript $i$ : | 3 | 2 | 1 | 0 |           |
|-----------------|---|---|---|---|-----------|
| Input carry     | 0 | 1 | 1 | 0 | $C_i$     |
| Augend          | 1 | 0 | 1 | 1 | $A_i$     |
| Addend          | 0 | 0 | 1 | 1 | $B_i$     |
| Sum             | 1 | 1 | 1 | 0 | $S_i$     |
| Output carry    | 0 | 0 | 1 | 1 | $C_{i+1}$ |

- The bits are added with full adders, starting from the least significant position (subscript 0), to form the sum bit and carry bit. The input carry  $C_0$  in the least significant position must be 0.
- The value of  $C_{i+1}$  in a given significant position is the output carry of the full adder. This value is transferred into the input carry of the full adder that adds the bits one higher significant position to the left.
- The sum bits are thus generated starting from the rightmost position and are available as soon as the corresponding previous carry bit is generated. All the carries must be generated for the correct sum bits to appear at the outputs.



Four Bit Binary Adder

#### HALF SUBTRACTOR:-

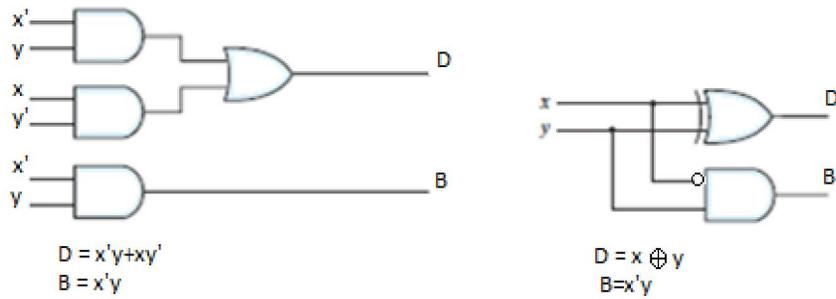
- This circuit needs two binary inputs and two binary outputs.
- Symbols  $x$  and  $y$  are assigned to the two inputs and  $D$  (for difference) and  $B$  (for borrow) to the outputs.
- The truth table for the half subtractor is listed in the below table.

| $x$ | $y$ | $D$ | $B$ |
|-----|-----|-----|-----|
| 0   | 0   | 0   | 0   |
| 0   | 1   | 1   | 1   |
| 1   | 0   | 1   | 0   |
| 1   | 1   | 0   | 0   |

Truth Table

- The  $B$  output is 1 only when the inputs are 0 and 1. The  $D$  output represents the least significant bit of the subtraction.
- The subtraction operation is done by using the following rules as
  - 0-0=0;
  - 0-1=1 with borrow 1;
  - 1-0=1;
  - 1-1=0.
- The simplified Boolean functions for the two outputs can be obtained directly from the truth table. The simplified sum-of-products expressions are

$$D = x'y + xy' \text{ and } B = x'y$$



- The logic diagram of the half adder implemented in sum of products is shown in the figure. It can be also implemented with an exclusive-OR and an AND gate with one inverted input.

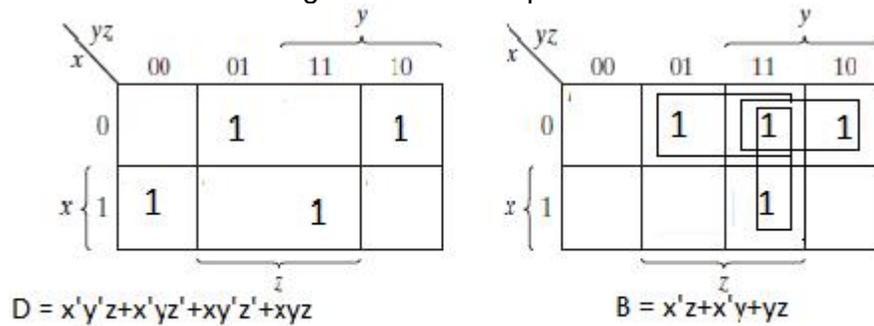
**FULL SUBTRACTOR:-**

- A full subtractor is a combinational circuit that forms the arithmetic subtraction operation of three bits.
- It consists of three inputs and two outputs. Two of the input variables, denoted by x and y, represent the two significant bits to be subtracted. The third input, z, is subtracted from the result of the first subtraction.

| x | y | z | D | B |
|---|---|---|---|---|
| 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 |

Truth Table

- Two outputs are necessary because the arithmetic subtraction of three binary digits ranges in value from 0 to 3, and binary representation of 2 or 3 needs two bits. The two outputs are designated by the symbols D for difference and B for borrow.
- The binary variable D gives the value of the least significant bit of the difference. The binary variable B gives the output borrow formed during the subtraction process.



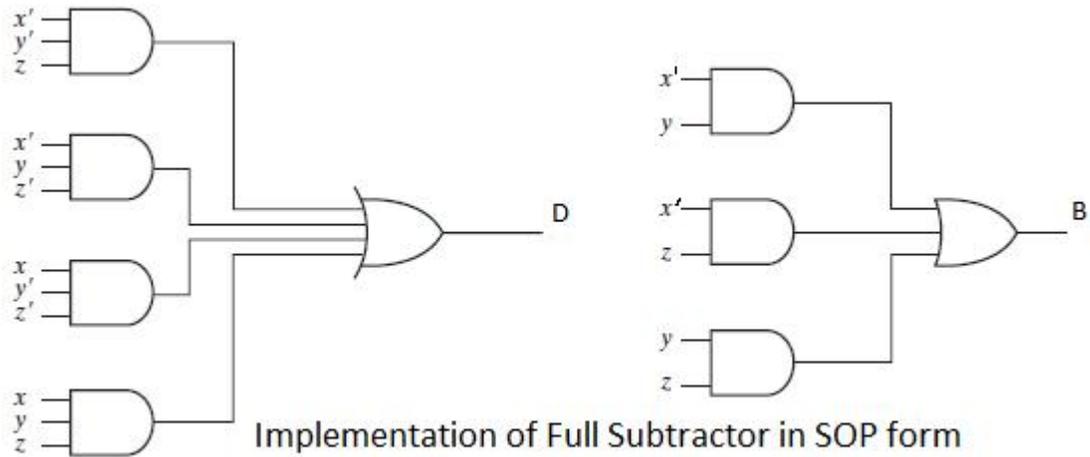
K-Map for full Subtractor

- The eight rows under the input variables designate all possible combinations of the three variables. The output variables are determined from the arithmetic subtraction of the input bits.
- The difference D becomes 1 when any one of the input is 1 or all three inputs are equal to 1 and the borrow B is 1 when the input combination is (0 0 1) or (0 1 0) or (0 1 1) or (1 1 1).
- The simplified expressions are

$$D = x'y'z + x'yz' + xy'z' + xyz$$

$$B = x'z + x'y + yz$$

- The logic diagram for the full adder implemented in sum-of-products form is shown in figure.



### MAGNITUDE COMPARATOR:-

- A magnitude comparator is a combinational circuit that compares two numbers A and B and determines their relative magnitudes.
- The following description is about a 2-bit magnitude comparator circuit.
- The outcome of the comparison is specified by three binary variables that indicate whether  $A < B$ ,  $A = B$ , or  $A > B$ .
- Consider two numbers, A and B, with two digits each. Now writing the coefficients of the numbers in descending order of significance:

$$A = A_1 A_0$$

$$B = B_1 B_0$$

- The two numbers are equal if all pairs of significant digits are equal i.e. if and only if  $A_1 = B_1$ , and  $A_0 = B_0$ .
- When the numbers are binary, the digits are either 1 or 0, and the equality of each pair of bits can be expressed logically with an exclusive-NOR function as

$$x_1 = A_1 B_1 + A_1' B_1'$$

$$\text{And } x_0 = A_0 B_0 + A_0' B_0'$$

- The equality of the two numbers A and B is displayed in a combinational circuit by an output binary variable that we designate by the symbol  $(A = B)$ .
- This binary variable is equal to 1 if the input numbers, A and B, are equal, and is equal to 0 otherwise.
- For equality to exist, all  $x_i$  variables must be equal to 1, a condition that dictates an AND operation of all variables:

$$(A = B) = x_1 x_0$$

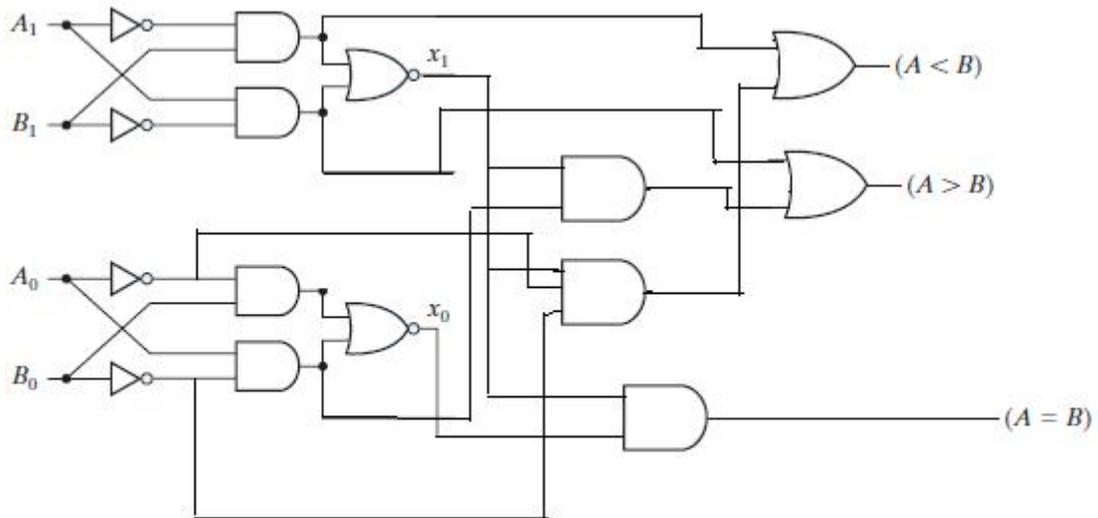
- The binary variable  $(A = B)$  is equal to 1 only if all pairs of digits of the two numbers are equal.
- To determine whether A is greater or less than B, we inspect the relative magnitudes of pairs of significant digits, starting from the most significant position. If the two digits of a pair are equal, we compare the next lower significant pair of digits. If the corresponding digit of A is 1 and that of B is 0, we conclude that  $A > B$ . If the corresponding digit of A is 0 and that of B is 1, we have  $A < B$ . The sequential comparison can be expressed logically by the two Boolean functions

$$(A > B) = A_1 B_1' + x_1 A_0 B_0'$$

$$(A < B) = A_1' B_1 + x_1 A_0' B_0'$$

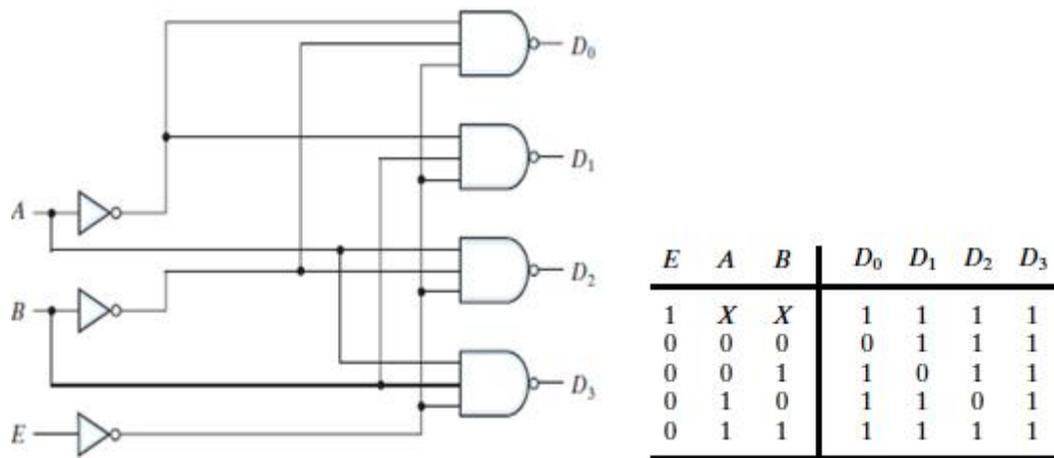
| A <sub>1</sub> | A <sub>0</sub> | B <sub>1</sub> | B <sub>0</sub> | A>B | A<B | A=B |
|----------------|----------------|----------------|----------------|-----|-----|-----|
| 0              | 0              | 0              | 0              | 0   | 0   | 1   |
| 0              | 0              | 0              | 1              | 0   | 1   | 0   |
| 0              | 0              | 1              | 0              | 0   | 1   | 0   |
| 0              | 0              | 1              | 1              | 0   | 1   | 0   |
| 0              | 1              | 0              | 0              | 1   | 0   | 0   |
| 0              | 1              | 0              | 1              | 0   | 0   | 1   |
| 0              | 1              | 1              | 0              | 0   | 1   | 0   |
| 0              | 1              | 1              | 1              | 0   | 1   | 0   |
| 1              | 0              | 0              | 0              | 1   | 0   | 0   |
| 1              | 0              | 0              | 1              | 1   | 0   | 0   |
| 1              | 0              | 1              | 0              | 0   | 0   | 1   |
| 1              | 0              | 1              | 1              | 0   | 1   | 0   |
| 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   | 0   | 1   |

Truth Table



Logic Diagram of 2-bit Magnitude Comparator

## DECODER:-



- A decoder is a combinational circuit that converts binary information from  $n$  input lines to a maximum of  $2^n$  unique output lines.
- If the  $n$ -bit coded information has unused combinations, the decoder may have fewer than  $2^n$  outputs.
- The decoders presented here are called  $n$ -to- $m$ -line decoders, where  $m \leq 2^n$ .
- Their purpose is to generate the  $2^n$  (or fewer) minterms of  $n$  input variables.
- Each combination of inputs will assert a unique output. The name decoder is also used in conjunction with other code converters, such as a BCD-to-seven-segment decoder.
- Consider the three-to-eight-line decoder circuit of three inputs are decoded into eight outputs, each representing one of the minterms of the three input variables.
- The three inverters provide the complement of the inputs, and each one of the eight AND gates generates one of the minterms.
- The input variables represent a binary number, and the outputs represent the eight digits of a number in the octal number system.
- However, a three-to-eight-line decoder can be used for decoding any three-bit code to provide eight outputs, one for each element of the code.
- A two-to-four-line decoder with an enable input constructed with NAND gates is shown in Fig.
- The circuit operates with complemented outputs and a complement enable input. The decoder is enabled when  $E$  is equal to 0 (i.e., active-low enable). As indicated by the truth table, only one output can be equal to 0 at any given time; all other outputs are equal to 1.
- The output whose value is equal to 0 represents the minterm selected by inputs  $A$  and  $B$ .
- The circuit is disabled when  $E$  is equal to 1, regardless of the values of the other two inputs.
- When the circuit is disabled, none of the outputs are equal to 0 and none of the minterms are selected.
- In general, a decoder may operate with complemented or un-complemented outputs.
- The enable input may be activated with a 0 or with a 1 signal.
- Some decoders have two or more enable inputs that must satisfy a given logic condition in order to enable the circuit.
- A decoder with enable input can function as a demultiplexer— a circuit that receives information from a single line and directs it to one of  $2^n$  possible output lines.
- The selection of a specific output is controlled by the bit combination of  $n$  selection lines.
- The decoder of Fig. can function as a one-to-four-line demultiplexer when  $E$  is taken as a data input line and  $A$  and  $B$  are taken as the selection inputs.
- The single input variable  $E$  has a path to all four outputs, but the input information is directed to only one of the output lines, as specified by the binary combination of the two selection lines  $A$  and  $B$ .
- This feature can be verified from the truth table of the circuit.
- For example, if the selection lines  $AB = 10$ , output  $D_2$  will be the same as the input value  $E$ , while all other outputs are maintained at 1.
- Since decoder and demultiplexer operations are obtained from the same circuit, a decoder with an enable input is referred to as a decoder – demultiplexer.
- A application of this decoder is binary-to-octal conversion.

### ENCODER:-

- An encoder is a digital circuit that performs the inverse operation of a decoder.
- An encoder has  $2n$  (or fewer) input lines and  $n$  output lines.
- The output lines, as an aggregate, generate the binary code corresponding to the input value.

| Inputs |       |       |       |       |       |       |       | Outputs |     |     |
|--------|-------|-------|-------|-------|-------|-------|-------|---------|-----|-----|
| $D_0$  | $D_1$ | $D_2$ | $D_3$ | $D_4$ | $D_5$ | $D_6$ | $D_7$ | $x$     | $y$ | $z$ |
| 1      | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0       | 0   | 0   |
| 0      | 1     | 0     | 0     | 0     | 0     | 0     | 0     | 0       | 0   | 1   |
| 0      | 0     | 1     | 0     | 0     | 0     | 0     | 0     | 0       | 1   | 0   |
| 0      | 0     | 0     | 1     | 0     | 0     | 0     | 0     | 0       | 1   | 1   |
| 0      | 0     | 0     | 0     | 1     | 0     | 0     | 0     | 1       | 0   | 0   |
| 0      | 0     | 0     | 0     | 0     | 1     | 0     | 0     | 1       | 0   | 1   |
| 0      | 0     | 0     | 0     | 0     | 0     | 1     | 0     | 1       | 1   | 0   |
| 0      | 0     | 0     | 0     | 0     | 0     | 0     | 1     | 1       | 1   | 1   |

- The above Encoder has eight inputs (one for each of the octal digits) and three outputs that generate the corresponding binary number.
- It is assumed that only one input has a value of 1 at any given time.
- The encoder can be implemented with OR gates whose inputs are determined directly from the truth table.
- Output  $z$  is equal to 1 when the input octal digit is 1, 3, 5, or 7.
- Output  $y$  is 1 for octal digits 2, 3, 6, or 7, and output  $x$  is 1 for digits 4, 5, 6, or 7.
- These conditions can be expressed by the following Boolean output functions:  
$$z = D_1 + D_3 + D_5 + D_7$$
$$y = D_2 + D_3 + D_6 + D_7$$
$$x = D_4 + D_5 + D_6 + D_7$$
- The encoder can be implemented with three OR gates.
- The encoder defined above has the limitation that only one input can be active at any given time.
- If two inputs are active simultaneously, the output produces an undefined combination.
- To resolve this ambiguity, encoder circuits must establish an input priority to ensure that only one input is encoded which is done in the Priority Encoder .

### PRIORITY ENCODER:-

- A priority encoder is an encoder circuit that includes the priority function.
- The operation of the priority encoder is such that if two or more inputs are equal to 1 at the same time, the input having the highest priority will take precedence.

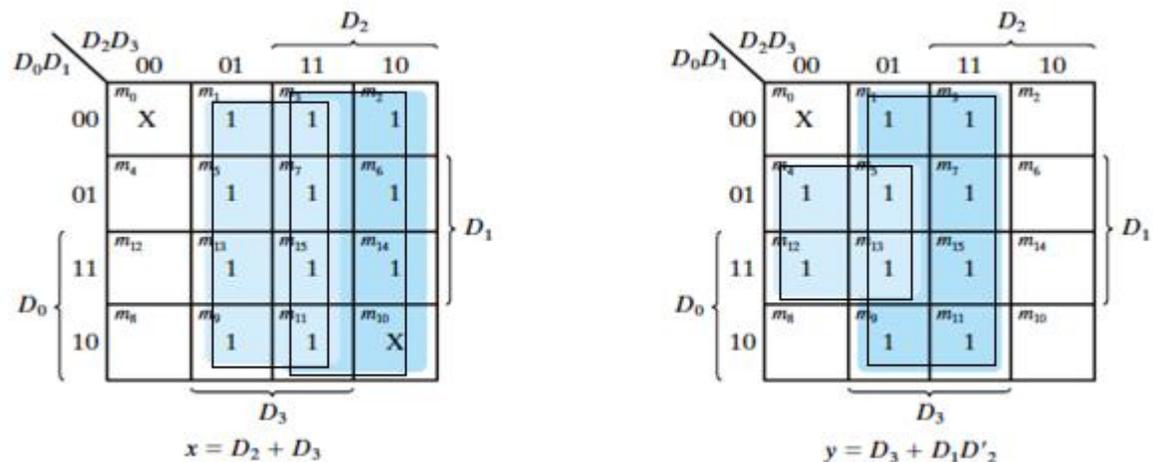
| Inputs |       |       |       | Outputs |     |     |
|--------|-------|-------|-------|---------|-----|-----|
| $D_0$  | $D_1$ | $D_2$ | $D_3$ | $x$     | $y$ | $V$ |
| 0      | 0     | 0     | 0     | X       | X   | 0   |
| 1      | 0     | 0     | 0     | 0       | 0   | 1   |
| X      | 1     | 0     | 0     | 0       | 1   | 1   |
| X      | X     | 1     | 0     | 1       | 0   | 1   |
| X      | X     | X     | 1     | 1       | 1   | 1   |

- In addition to the two outputs  $x$  and  $y$  , the circuit has a third output designated by  $V$  ; this is a valid bit indicator that is set to 1 when one or more inputs are equal to 1.

- If all inputs are 0, there is no valid input and V is equal to 0.
- The other two outputs are not inspected when V equals 0 and are specified as don't-care conditions.
- Here X 's in output columns represent don't-care conditions, the X 's in the input columns are useful for representing a truth table in condensed form.

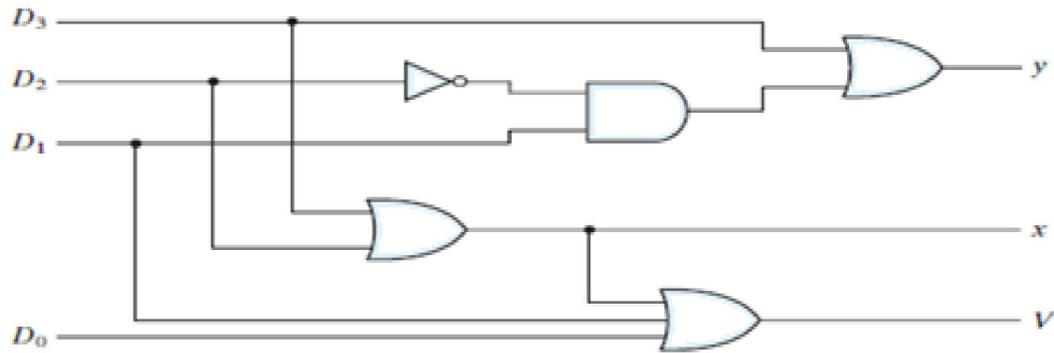
| Inputs |       |       |       | Outputs |     |     |
|--------|-------|-------|-------|---------|-----|-----|
| $D_0$  | $D_1$ | $D_2$ | $D_3$ | $x$     | $y$ | $V$ |
| 0      | 0     | 0     | 0     | X       | X   | 0   |
| 1      | 0     | 0     | 0     | 0       | 0   | 1   |
| X      | 1     | 0     | 0     | 0       | 1   | 1   |
| X      | X     | 1     | 0     | 1       | 0   | 1   |
| X      | X     | X     | 1     | 1       | 1   | 1   |

- Higher the subscript number, the higher the priority of the input.
- Input D3 has the highest priority, so, regardless of the values of the other inputs, when this input is 1, the output for xy is 11 (binary 3).
- If D2 = 1, provided that D3 = 0, regardless of the values of the other two lower priority inputs the output is 10.
- The output for D1 is generated only if higher priority inputs are 0, and so on down the priority levels.



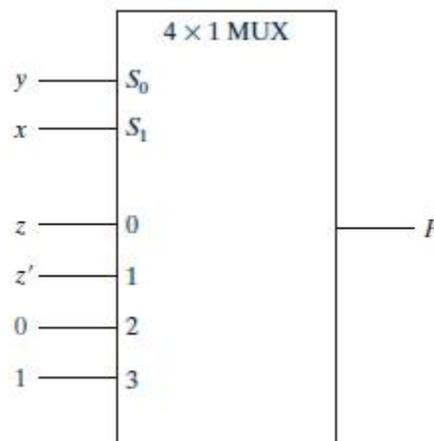
- The maps for simplifying outputs x and y are shown in above Fig.
- The minterms for the two functions are derived from its truth table.
- Although the table has only five rows, when each X in a row is replaced first by 0 and then by 1, we obtain all 16 possible input combinations.
- For example, the fourth row in the table, with inputs XX10, represents the four minterms 0010, 0110, 1010, and 1110. The simplified Boolean expressions for the priority encoder are obtained from the maps.
- The condition for output V is an OR function of all the input variables.
- The priority encoder is implemented according to the following Boolean functions:

$$\begin{aligned}
 x &= D_2 + D_3 \\
 y &= D_3 + D_1 D'_2 \\
 V &= D_0 + D_1 + D_2 + D_3
 \end{aligned}$$

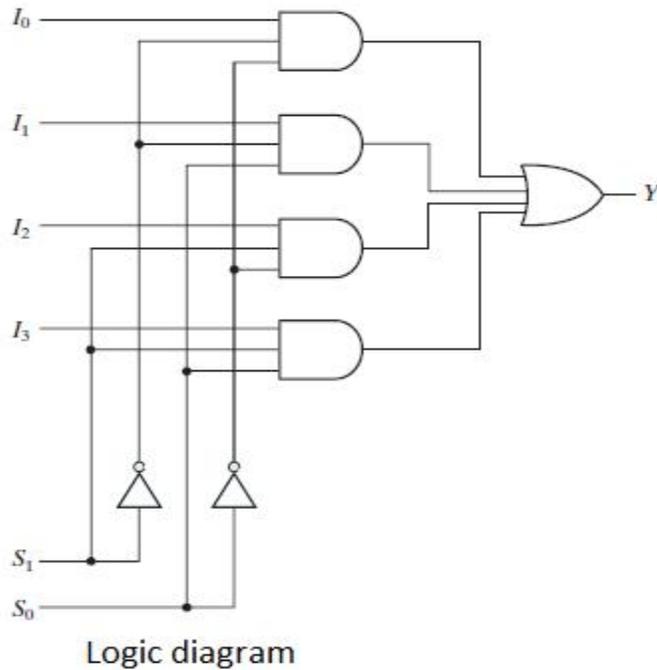


**MULTIPLEXER:-**

- A multiplexer is a combinational circuit that selects binary information from one of many input lines and directs it to a single output line.
- The selection of a particular input line is controlled by a set of selection lines.
- Normally, there are  $2^n$  input lines and n selection lines whose bit combinations determine which input is selected.
- A four-to-one-line multiplexer is shown in the below figure. Each of the four inputs,  $I_0$  through  $I_3$ , is applied to one input of an AND gate.
- Selection lines  $S_1$  and  $S_0$  are decoded to select a particular AND gate. The outputs of the AND gates are applied to a single OR gate that provides the one-line output.
- The function table lists the input that is passed to the output for each combination of the binary selection values.
- To demonstrate the operation of the circuit, consider the case when  $S_1S_0 = 10$ .
- The AND gate associated with input  $I_2$  has two of its inputs equal to 1 and the third input connected to  $I_2$ .
- The other three AND gates have at least one input equal to 0, which makes their outputs equal to 0. The output of the OR gate is now equal to the value of  $I_2$ , providing a path from the selected input to the output.
- A multiplexer is also called a data selector, since it selects one of many inputs and steers the binary information to the output line.



(b) Multiplexer implementation



| S <sub>1</sub> | S <sub>0</sub> | Y              |
|----------------|----------------|----------------|
| 0              | 0              | I <sub>0</sub> |
| 0              | 1              | I <sub>1</sub> |
| 1              | 0              | I <sub>2</sub> |
| 1              | 1              | I <sub>3</sub> |

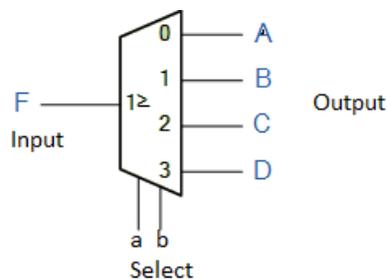
Truth table

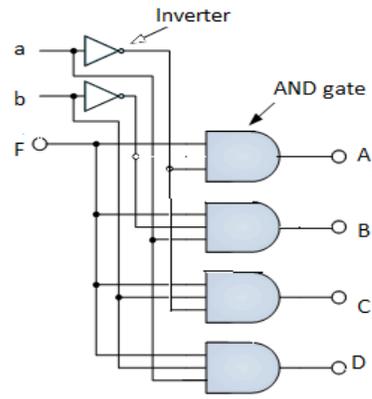
**DEMULTIPLEXER:-**

- The data distributor, known more commonly as a Demultiplexer or “Demux” for short, is the exact opposite of the Multiplexer.
- The demultiplexer takes one single input data line and then switches it to any one of a number of individual output lines one at a time. The demultiplexer converts a serial data signal at the input to a parallel data at its output lines as shown below.
- The Boolean expression for this 1-to-4 demultiplexer above with outputs A to D and data select lines a, b is given as:

$$F = (ab)'A + a'bB + ab'C + abD$$

- The function of the demultiplexer 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” as shown.





Logic Diagram

- Unlike multiplexers which convert data from a single data line to multiple lines and demultiplexers which convert multiple lines to a single data line, there are devices available which convert data to and from multiple lines and in the next tutorial about combinational logic devices.
- Standard demultiplexer IC packages available are the TTL 74LS138 1 to 8-output demultiplexer, the TTL 74LS139 Dual 1-to-4 output demultiplexer or the CMOS CD4514 1-to-16 output demultiplexer.

| Output Select |   | Data output Selected |
|---------------|---|----------------------|
| b             | a |                      |
| 0             | 0 | A                    |
| 0             | 1 | B                    |
| 1             | 0 | C                    |
| 1             | 1 | D                    |

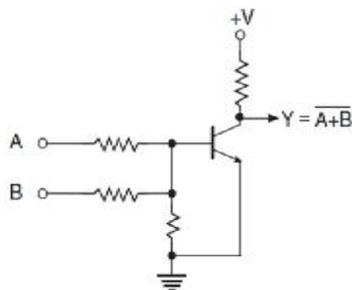
Truth Table

# LOGIC FAMILIES

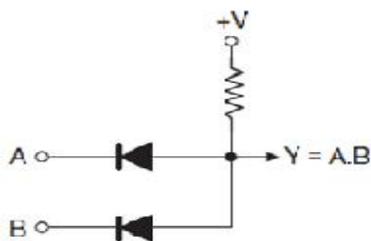
- A circuit configuration or approach used to produce a type of digital integrated circuit is called Logic Family.
- By using logic families we can generate different logic functions, when fabricated in the form of an IC with the same approach, or in other words belonging to the same logic family, will have identical electrical characteristics.
- The set of digital ICs belonging to the same logic family are electrically compatible with each other.
- Some common Characteristics of the Same Logic Family include Supply voltage range, speed of response, power dissipation, input and output logic levels, current sourcing and sinking capability, fan-out, noise margin, etc.
- Choosing digital ICs from the same logic family guarantees that these ICs are compatible with respect to each other and that the system as a whole performs the intended logic function.

## TYPES OF LOGIC FAMILY:-

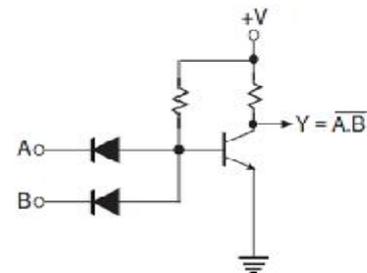
- The entire range of digital ICs is fabricated using either bipolar devices or MOS devices or a combination of the two.
- Bipolar families include:-
  - Diode logic (DL)
  - Resistor-Transistor logic (RTL)
  - Diode-transistor logic (DTL)
  - Transistor- Transistor logic (TTL)
  - Emitter Coupled Logic (ECL),  
(also known as Current Mode Logic(CML))
  - Integrated Injection logic (I<sup>2</sup>L)
- The Bi-MOS logic family uses both bipolar and MOS devices.



Resistor -Transistor  
logic (RTL)



Diode Logic  
(DL)



Diode-Transistor logic  
(DTL)

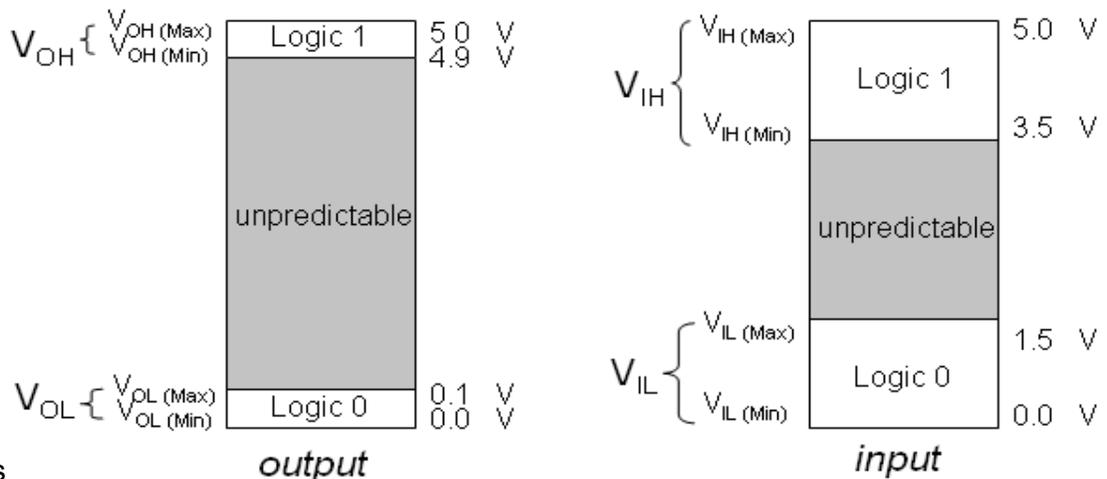
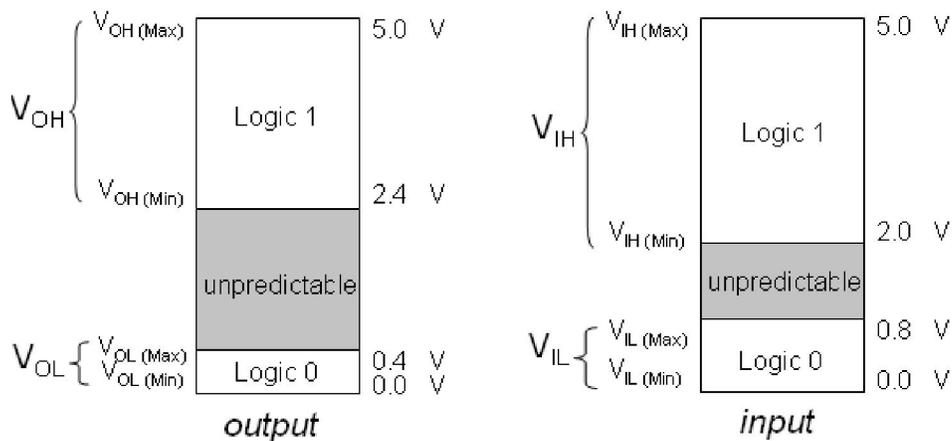
- Above are some example of DL, RTL and DTL.
- MOS families include:-
  - The PMOS family (using P-channel MOSFETs)
  - The NMOS family (using N-channel MOSFETs)
  - The CMOS family (using both N- and P-channel devices)

## SOME OPERATIONAL PROPERTIES OF LOGIC FAMILY:-

### DC Supply Voltage:-

- The nominal value of the dc supply voltage for TTL (transistor-transistor logic) and CMOS (complementary metal-oxide semiconductor) devices is +5V. Although omitted from logic diagrams for simplicity, this voltage is connected to V<sub>cc</sub> or V<sub>DD</sub> pin of an IC package and ground is connected to the GND pin.

## TTL Logic Levels



## CMOS Logic Levels

### Noise Immunity:-

- Noise is the unwanted voltage that is induced in electrical circuits and can present a threat to the poor operation of the circuit. In order not to be adversely effected by noise, a logic circuit must have a certain amount of 'noise immunity'.
- This is the ability to tolerate a certain amount of unwanted voltage fluctuation on its inputs without changing its output state is called Noise Immunity.

### Noise Margin:-

- A measure of a circuit's noise immunity is called 'noise margin' which is expressed in volts.
- There are two values of noise margin specified for a given logic circuit: the HIGH ( $V_{NH}$ ) and LOW ( $V_{NL}$ ) noise margins.

These are defined by following equations :

$$V_{NH} = V_{OH} (\text{Min}) - V_{IH} (\text{Min}) \quad V_{NL} = V_{IL} (\text{Max}) - V_{OL} (\text{Max})$$

### Power Dissipation:-

- A logic gate draws  $IC_{CH}$  current from the supply when the gate is in the HIGH output state, draws  $IC_{CL}$  current from the supply in the LOW output state.
- Average power is  

$$PD = V_{CC} ICC \text{ where } ICC = (IC_{CH} + IC_{CL}) / 2$$

### Propagation Delay time:-

- When a signal passes ( propagates ) through a logic circuit, it always experiences a time delay as shown below. A change in the output level always occurs a short time, called 'propagation delay time' , later than the change in the input level that caused it.

### Fan Out of Gates:-

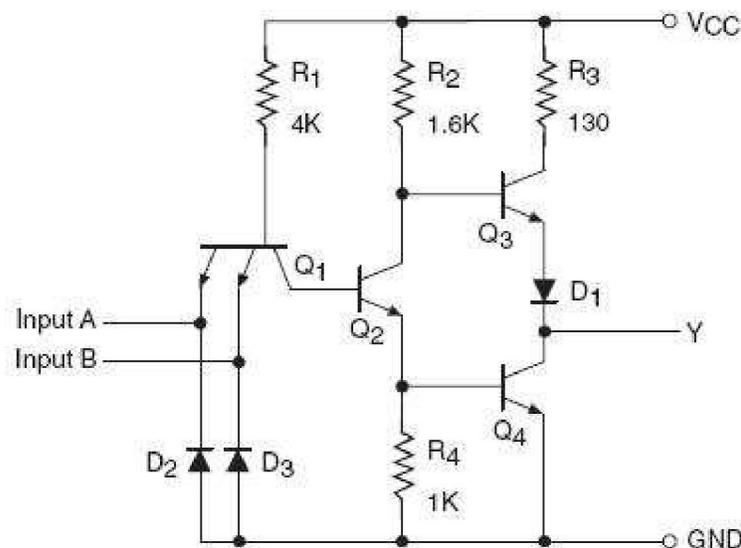
- When the output of a logic gate is connected to one or more inputs of other gates, a load on the driving gate is created. There is a limit to the number of load gates that a given gate can drive. This limit is called the 'Fan-Out' of the gate.

### TRANSISTOR-TRANSISTOR LOGIC:-

- In Transistor-Transistor logic or just TTL, logic gates are built only around transistors.
- TTL was developed in 1965. Through the years basic TTL has been improved to meet performance requirements. There are many versions or families of TTL.
- For example
  - Standard TTL
  - High Speed TTL (twice as fast, twice as much power)
  - Low Power TTL (1/10 the speed, 1/10 the power of "standard" TTL)
  - Schottky TTL etc. (for high-frequency uses )
- All TTL logic families have three configurations for outputs
  1. Totem pole output
  2. Open collector output
  3. Tristate output

### Totem pole output:-

- Addition of an active pull up circuit in the output of a gate is called totem pole.
- To increase the switching speed of the gate which is limited due to the parasitic capacitance at the output totem pole is used.
- The circuit of a totem-pole NAND gate is shown below, which has got three stages
  1. Input Stage
  2. Phase Splitter Stage
  3. Output Stage



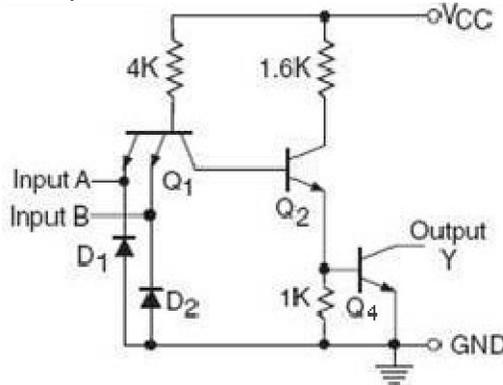
- Transistor Q1 is a two-emitter NPN transistor, which is equivalent to two NPN transistors with their base and emitter terminals tied together.
- The two emitters are the two inputs of the NAND gate
  - In TTL technology multiple emitter transistors are used for the input devices
  - Diodes D2 and D3 are protection diodes used to limit negative input voltages.
- When there is large negative voltage at input, the diode conducts and shorts it to the ground Q2 provides complementary voltages for the output transistors Q3 and Q4.
- The combination of Q3 and Q4 forms the output circuit often referred to as a totem pole arrangement (Q4 is stacked on top of Q3). In such an arrangement, either Q3 or Q4 conducts at a time depending upon the logic status of the inputs
  - Diode D1 ensures that Q4 will turn off when Q2 is on (HIGH input)
  - The output Y is taken from the top of Q3

Advantages of Totem Pole Output:-

- The features of this arrangement are
  1. Low power consumption
  2. Fast switching
  3. Low output impedance

OPEN COLLECTOR OUTPUT:-

- Figure below shows the circuit of a typical TTL gate with open-collector output. Observe here that the circuit elements associated with Q3 in the totem-pole circuit are missing and the collector of Q4 is left open-circuited, hence the name open-collector.



- An open-collector output can present a logic LOW output. Since there is no internal path from the output Y to the supply voltage  $V_{CC}$ , the circuit cannot present a logic HIGH on its own.

Advantages of Open Collector Outputs:-

- Open-collector outputs can be tied directly together which results in the logical ANDing of the outputs. Thus the equivalent of an AND gate can be formed by simply connecting the outputs.
- Increased current levels - Standard TTL gates with totem-pole outputs can only provide a HIGH current output of 0.4 mA and a LOW current of 1.6 mA. Many open-collector gates have increased current ratings.
- Different voltage levels - A wide variety of output HIGH voltages can be achieved using open-collector gates. This is useful in interfacing different logic families that have different voltage and current level requirements.

Disadvantage of open-collector gates:-

- They have slow switching speed. This is because the value of pull-up resistor is in kW, which results in a relatively long time Constants

Comparison of Totem Pole and Open Collector Output:-

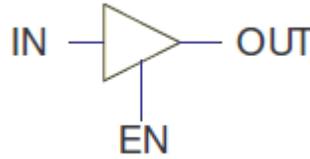
- The major advantage of using a totem-pole connection is that it offers low-output impedance in both the HIGH and LOW output states

| Totem Pole   | Open Collector   |
|--|--|
| Output stage consists of pull-up transistor (Q3), diode resistor and pull-down transistor (Q4) | Output stage consists of only pull-down transistor                 |
| External pull-up resistor is not required  | External pull-up resistor is required for proper operation of gate |
| Output of two gates cannot be tied together  | Output of two gates can be tied together using wired AND technique |
| Operating speed is high  | Operating speed is low   |

TRISTATE (THREE-STATE) LOGIC OUPUT:-

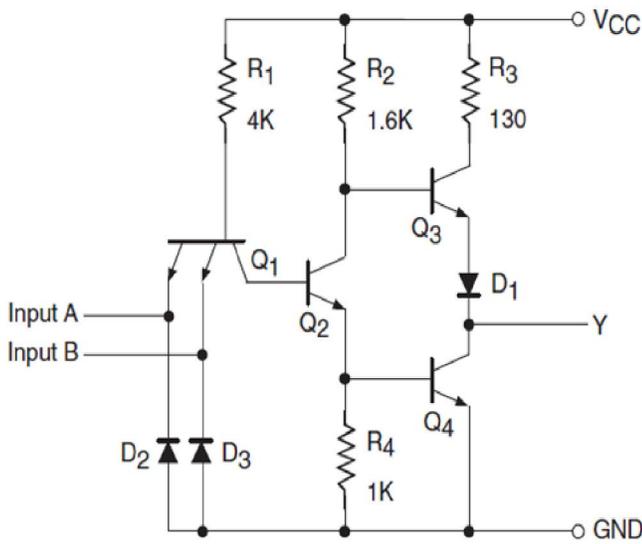
- Tristate output combines the advantages of the totem-pole and open collector circuits.
- Three output states are HIGH, LOW, and high impedance (Hi-Z).

| EN | IN | OUT  |
|----|----|------|
| 0  | X  | HI-Z |
| 1  | 0  | 0    |
| 1  | 1  | 1    |



- For the symbol and truth table, IN is the data input, and EN, the additional enable input for control. For EN = 0, regardless of the value on IN (denoted by X), the output value is Hi-Z. For EN = 1, the output value follows the input value.
- Data input, IN, can be inverted. Control input, EN, can be inverted by addition of "bubbles" to signals IN OUT EN.
- This requires two inputs: input and enable EN is to make output Hi-Z or follow input.

STANDARD TTL NAND GATE:



| A | B | Q1   | Q2  | Q3  | Q4   | Y |
|---|---|------|-----|-----|------|---|
| L | L | sat  | off | off | Of f | H |
| L | H | sat  | off | off | Of f | H |
| H | L | sat  | off | off | Of f | H |
| H | H | ia m | sat | sat | On   | L |

CMOS TECHNOLOGY:-

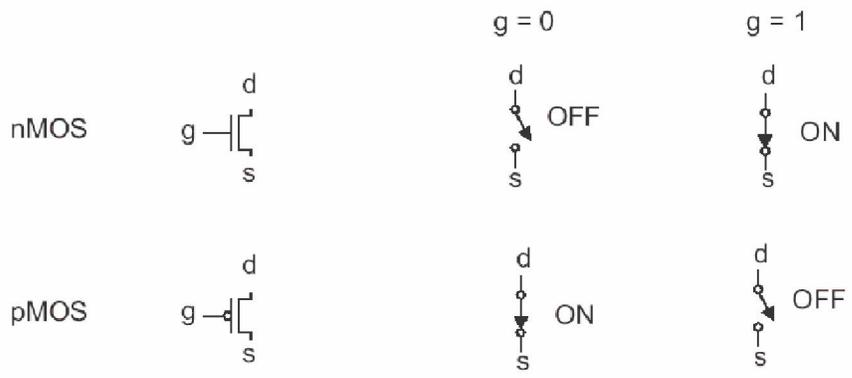
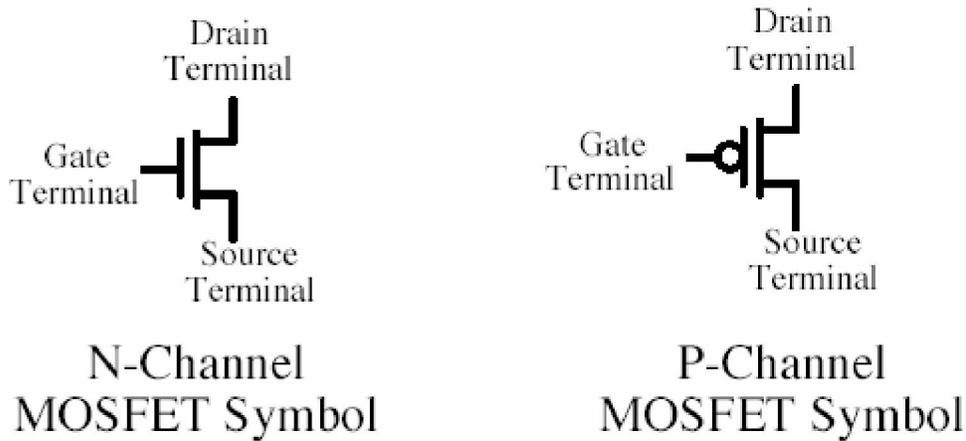
- MOS stands for Metal Oxide Semiconductor and this technology uses FETs.
- MOS can be classified into three sub-families:
  - PMOS (P-channel)
  - NMOS (N-channel)
  - CMOS (Complementary MOS, most common)
- The following simplified symbols are used to represent MOSFET transistors in most CMOS. The gate of a MOS transistor controls the flow of the current between the drain and the source. The MOS transistor can be viewed as a simple ON/OFF switch.

Advantages of MOS Digital ICs:-

- They are simple and inexpensive to fabricate.
- Can be used for Higher integration and consume little power.

Disadvantages of MOS Digital ICs:-

- There is possibility for Static-electricity damage.
- They are slower than TTL.

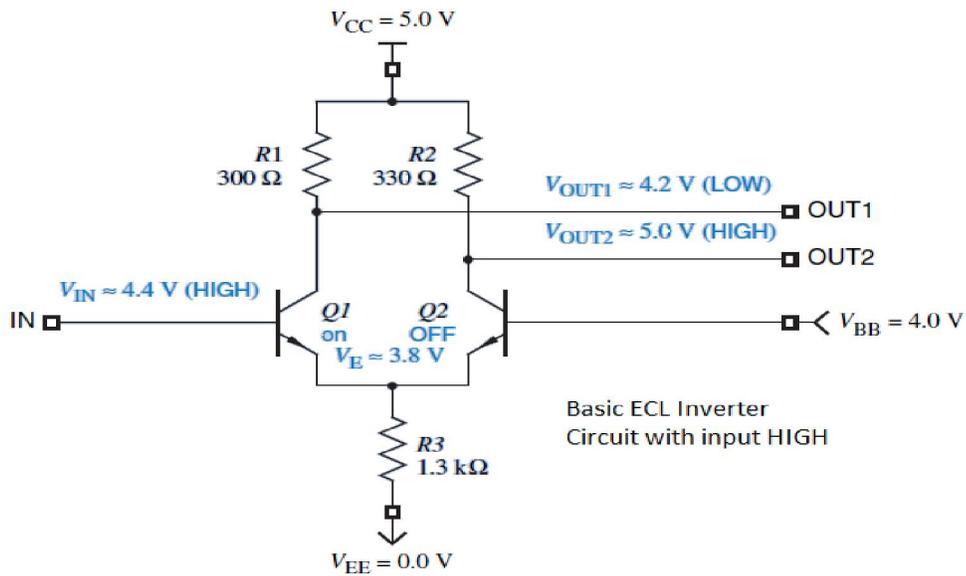


**ECL: EMITTER-COUPLED LOGIC:-**

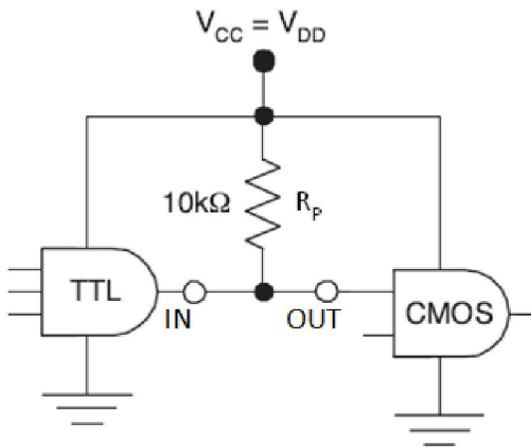
- The key to reduce propagation delay in a bipolar logic family is to prevent a gate's transistors from saturating. It is possible to prevent saturation by using a radically different circuit structure, called current-mode logic (CML) or emitter-coupled logic (ECL).
- Unlike the other logic families in this chapter, ECL does not produce a large voltage swing between the LOW and HIGH levels but it has a small voltage swing, less than a volt, and it internally switches current between two possible paths, depending on the output state.

**Basic ECL Circuit**

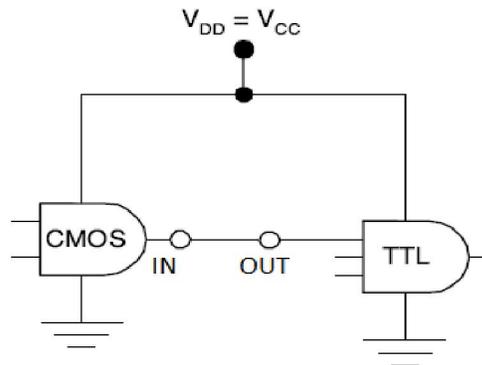
- The basic idea of current-mode logic is illustrated by the inverter/buffer circuit in the figure. This circuit has both an inverting output (OUT1) and a non-inverting output (OUT2).
- Two transistors are connected as a differential amplifier with a common emitter resistor.
- The supply voltages for this example are  $V_{CC} = 5.0$ ,  $V_{BB} = 4.0$ , and  $V_{EE} = 0$  V, and the input LOW and HIGH levels are defined to be 3.6 and 4.4 V. This circuit actually produces output LOW and HIGH levels that are 0.6 V higher (4.2 and 5.0 V).



### INTERFACING OF TTL TO CMOS



### INTERFACING OF CMOS TO TTL



### TTL vs. CMOS:-

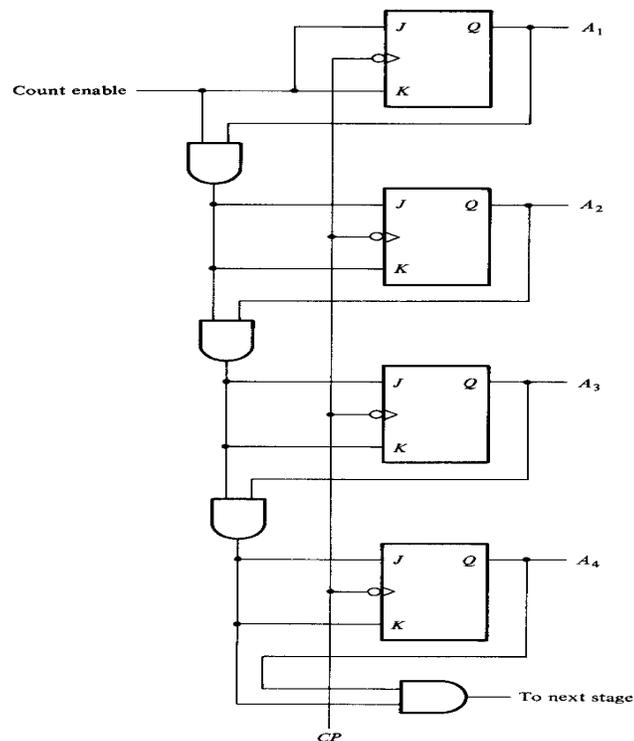
- TTL has less propagation delay than CMOS i.e. TTL is good where high speed is needed.
- And CMOS 4000 is good for Battery equipment and where speed is not so important.
- CMOS requires less power than TTL i.e. power dissipation and hence power consumption is less for CMOS.

# COUNTER

- A counter is a device which stores (and sometimes displays) the number of times a particular event or process has occurred. In electronics, counters can be implemented quite easily using register-type circuits.
- There are different types of counters, viz.
  - Asynchronous (ripple) counter
  - Synchronous counter
  - Decade counter
  - Up/down counter
  - Ring counter
  - Johnson counter
  - Cascaded counter
  - Modulus counter.

## Synchronous counter

- A 4-bit synchronous counter using JK flip-flops is shown in the figure.
- In synchronous counters, the clock inputs of all the flip-flops are connected together and are triggered by the input pulses. Thus, all the flip-flops change state simultaneously (in parallel).

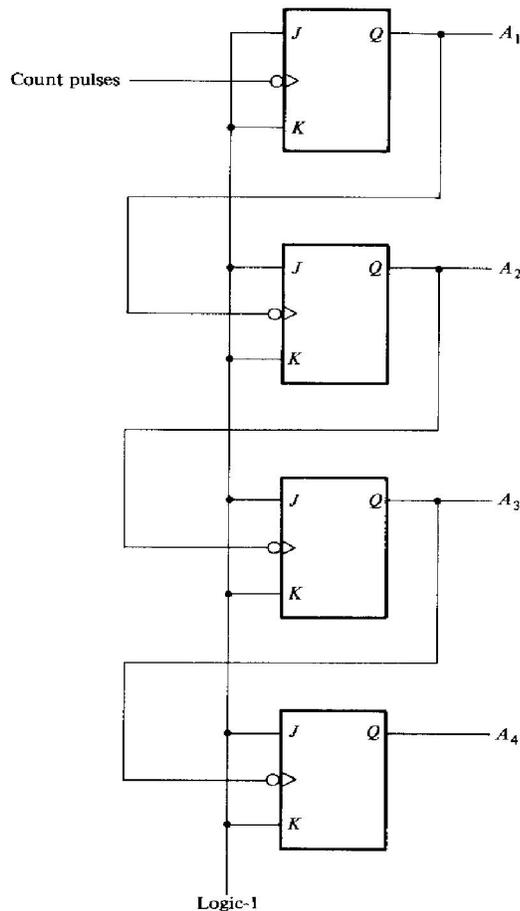


- The circuit below is a 4-bit synchronous counter.
- The J and K inputs of FF0 are connected to HIGH. FF1 has its J and K inputs connected to the output of FF0, and the J and K inputs of FF2 are connected to the output of an AND gate that is fed by the outputs of FF0 and FF1.
- A simple way of implementing the logic for each bit of an ascending counter (which is what is depicted in the image to the right) is for each bit to toggle when all of the less significant bits are at a logic high state.
- For example, bit 1 toggles when bit 0 is logic high; bit 2 toggles when both bit 1 and bit 0 are logic high; bit 3 toggles when bit 2, bit 1 and bit 0 are all high; and so on.

- Synchronous counters can also be implemented with hardware finite state machines, which are more complex but allow for smoother, more stable transitions.

## Asynchronous Counter

- An asynchronous (ripple) counter is a single d-type flip-flop, with its J (data) input fed from its own inverted output.
- This circuit can store one bit, and hence can count from zero to one before it overflows (starts over from 0).



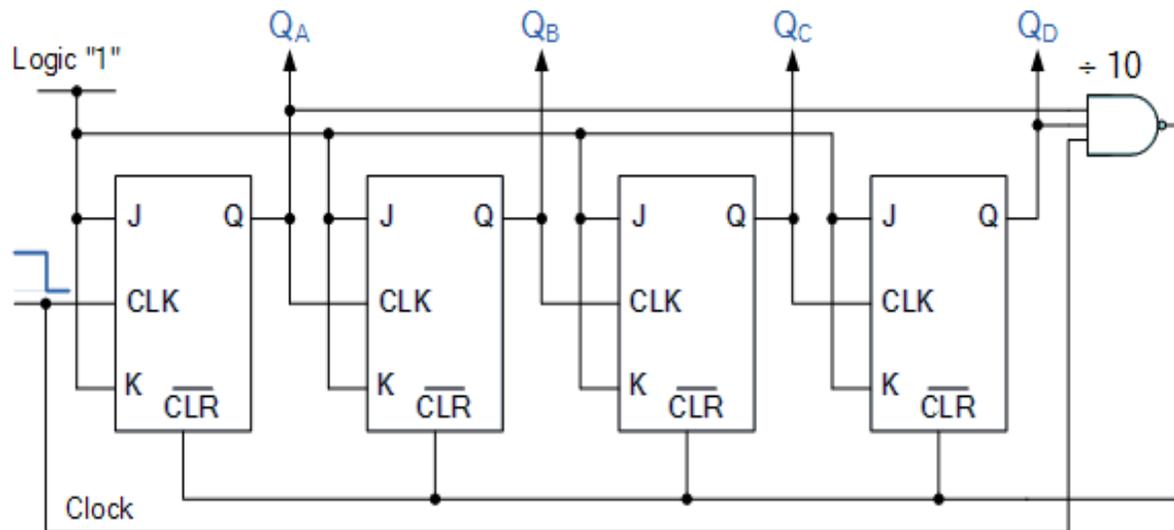
- This counter will increment once for every clock cycle and takes two clock cycles to overflow, so every cycle it will alternate between a transition from 0 to 1 and a transition from 1 to 0.
- This creates a new clock with a 50% duty cycle at exactly half the frequency of the input clock.
- If this output is then used as the clock signal for a similarly arranged D flip-flop, remembering to invert the output to the input, one will get another 1 bit counter that counts half as fast. These together yield a two-bit counter.
- Additional flip-flops can be added, by always inverting the output to its own input, and using the output from the previous flip-flop as the clock signal. The result is called a ripple counter, which can count to  $2^n - 1$ , where  $n$  is the number of bits (flip-flop stages) in the counter.
- Ripple counters suffer from unstable outputs as the overflows "ripple" from stage to stage, but they find application as dividers for clock signals.

## Modulus Counter

- A modulus counter is that which produces an output pulse after a certain number of input pulses is applied.
- In modulus counter the total count possible is based on the number of stages, i.e., digit positions.

- Modulus counters are used in digital computers.
- A binary modulo-8 counter with three flip-flops, i.e., three stages, will produce an output pulse, i.e., display an output one-digit, after eight input pulses have been counted, i.e., entered or applied. This assumes that the counter started in the zero-condition.

## Asynchronous Decade Counter



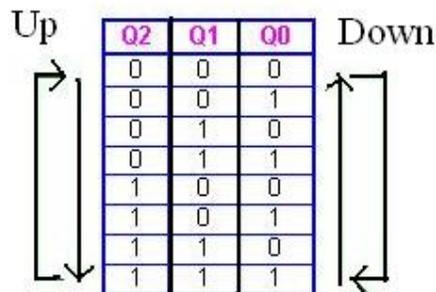
- A decade counter can count from BCD "0" to BCD "9".
- A decade counter requires resetting to zero when the output count reaches the decimal value of 10, i.e. when DCBA = 1010 and this condition is fed back to the reset input.
- A counter with a count sequence from binary "0000" (BCD = "0") through to "1001" (BCD = "9") is generally referred to as a BCD binary-coded-decimal counter because its ten state sequence is that of a BCD code but binary decade counters are more common.
- This type of asynchronous counter counts upwards on each leading edge of the input clock signal starting from 0000 until it reaches an output 1001 (decimal 9).
- Both outputs  $Q_A$  and  $Q_D$  are now equal to logic "1" and the output from the NAND gate changes state from logic "1" to a logic "0" level and whose output is also connected to the CLEAR ( CLR ) inputs of all the J-K Flip-flops.
- This signal causes all of the Q outputs to be reset back to binary 0000 on the count of 10. Once  $Q_A$  and  $Q_D$  are both equal to logic "0" the output of the NAND gate returns back to a logic level "1" and the counter restarts again from 0000. We now have a decade or Modulo-10 counter.

## Decade Counter Truth Table

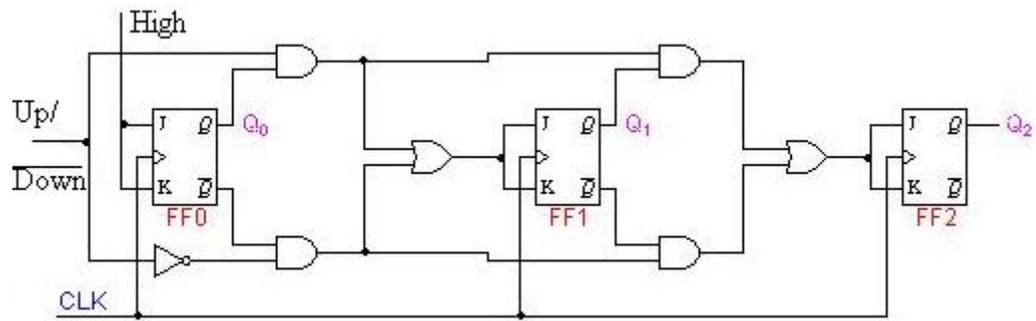
| Clock Count | Output bit Pattern                      |    |    |    | Decimal Value |
|-------------|---|----|----|----|---------------|
|             | QD                                      | QC | QB | QA |               |
| 1           | 0                                       | 0  | 0  | 0  | 0             |
| 2           | 0                                       | 0  | 0  | 1  | 1             |
| 3           | 0                                       | 0  | 1  | 0  | 2             |
| 4           | 0                                       | 0  | 1  | 1  | 3             |
| 5           | 0                                       | 1  | 0  | 0  | 4             |
| 6           | 0                                       | 1  | 0  | 1  | 5             |
| 7           | 0                                       | 1  | 1  | 0  | 6             |
| 8           | 0                                       | 1  | 1  | 1  | 7             |
| 9           | 1                                       | 0  | 0  | 0  | 8             |
| 10          | 1                                       | 0  | 0  | 1  | 9             |
| 11          | Counter Resets its Outputs back to Zero |    |    |    |               |

## Up/Down Counter

- In a synchronous up-down binary counter the flip-flop in the lowest-order position is complemented with every pulse.
- A flip-flop in any other position is complemented with a pulse, provided all the lower-order pulse equal to 0.
- Up/Down counter is used to control the direction of the counter through a certain sequence.



- From the sequence table we can observe that:
  - For both the UP and DOWN sequences,  $Q_0$  toggles on each clock pulse.
  - For the UP sequence,  $Q_1$  changes state on the next clock pulse when  $Q_0=1$ .
  - For the DOWN sequence,  $Q_1$  changes state on the next clock pulse when  $Q_0=0$ .
  - For the UP sequence,  $Q_2$  changes state on the next clock pulse when  $Q_0=Q_1=1$ .
  - For the DOWN sequence,  $Q_2$  changes state on the next clock pulse when  $Q_0=Q_1=0$ .



- These characteristics are implemented with the AND, OR & NOT logic connected as shown in the logic diagram above.

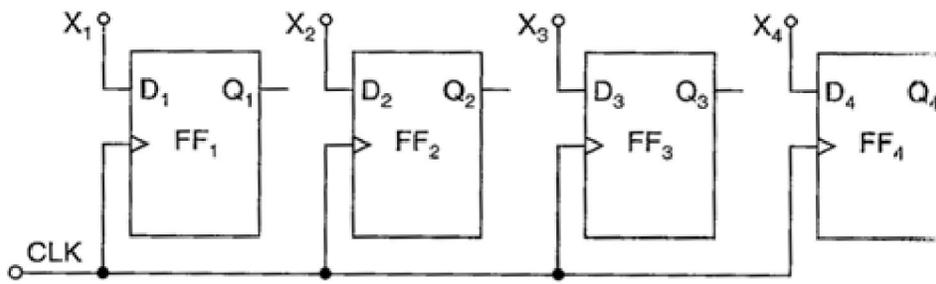
# REGISTERS

## INTRODUCTION:-

- The sequential circuits known as register are very important logical block in most of the digital systems.
- Registers are used for storage and transfer of binary information in a digital system.
- A register is mostly used for the purpose of storing and shifting binary data entered into it from an external source and has no characteristics internal sequence of states.
- The storage capacity of a register is defined as the number of bits of digital data, it can store or retain.
- These registers are normally used for temporary storage of data.

## BUFFER REGISTER:-

- These are the simplest registers and are used for simply storing a binary word.
- These may be controlled by Controlled Buffer Register.
- D flip – flops are used for constructing a buffer register or other flip- flop can be used.
- The figure shown below is a 4- bit buffer register.



Logic diagram of a 4-bit buffer register.

- The binary word to be stored is applied to the data terminals.
- When the clock pulse is applied, the output word becomes the same as the word applied at the input terminals, i.e. the input word is loaded into the register by the application of clock pulse.
- When the positive clock edge arrives, the stored word becomes:

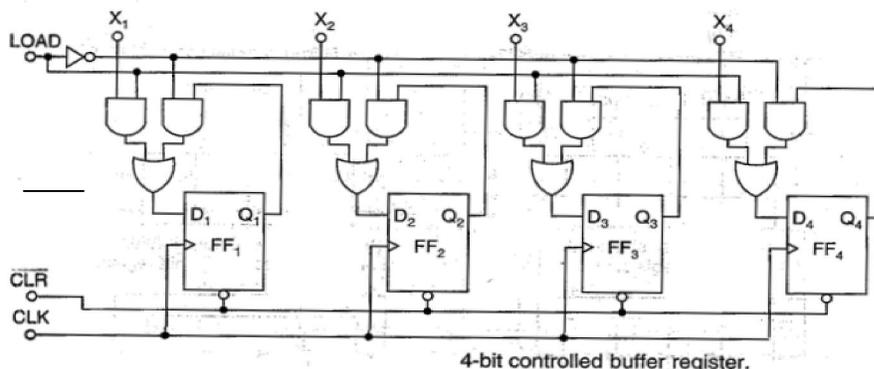
$$Q_4 Q_3 Q_2 Q_1 = X_4 X_3 X_2 X_1$$

$$\text{or } Q = X .$$

This circuit is too primitive to be of any use.

## CONTROLLED BUFFER REGISTER:-

- The figure shows a controlled buffer register.



4-bit controlled buffer register.

- If  $\overline{\text{CLR}}$  goes LOW, all the flip-flops are RESET and the output becomes,  $Q = 0000$ .
- When  $\overline{\text{CLR}}$  is HIGH, the register is ready for action

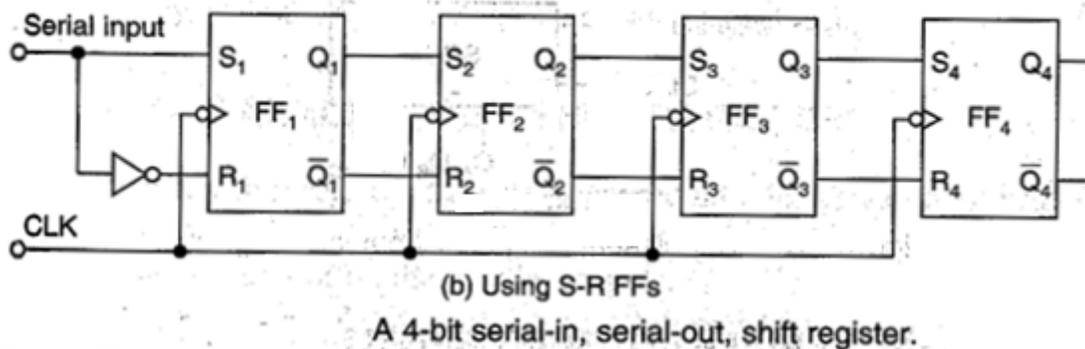
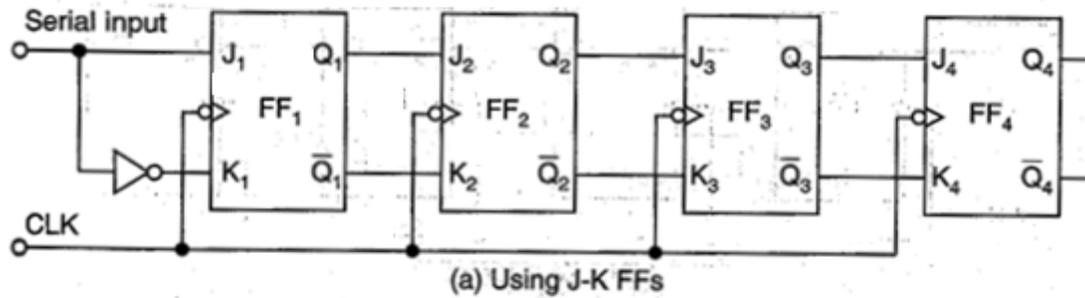
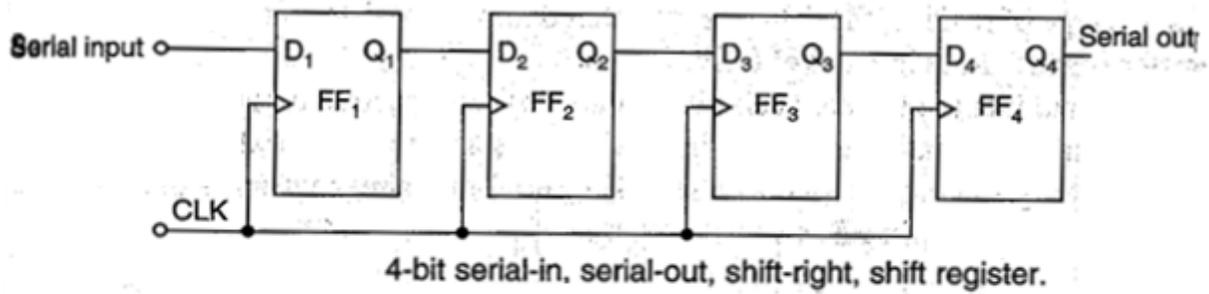
- LOAD is control input.
- When LOAD is HIGH, the data bits X can reach the D inputs of FFs.
- At the positive going edge of the next clock pulse, the register is loaded, i.e.
 
$$Q_4 Q_3 Q_2 Q_1 = X_4 X_3 X_2 X_1$$
 or  $Q = X$  .
- When LOAD is LOW, the X bits cannot reach the FFs. At the same time the inverted signal LOAD is HIGH. This forces each flip-flop output to feedback to its data input.
- Therefore data is circulated or retained as each clock pulse arrives.
- In other words the content register remains unchanged in spite of the clock pulses.
- Longer buffer registers can be built by adding more FFs.

### **CONTROLLED BUFFER REGISTER:-**

- A number of FFs connected together such that data may be shifted into and shifted out of them is called a shift register.
- Data may be shifted into or out of the register either in serial form or in parallel form.
- There are four basic types of shift registers
  1. Serial in, serial out
  2. Serial in, parallel out
  3. Parallel in, serial out
  4. Parallel in , parallel out

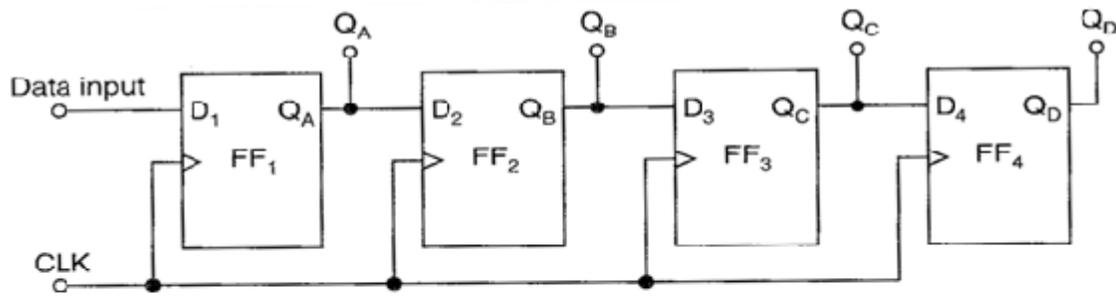
### **SERIAL IN, SERIAL OUT SHIFT REGISTER:-**

- This type of shift register accepts data serially, i.e., one bit at a time and also outputs data serially.
- The logic diagram of a four bit serial in, serial out shift register is shown in below figure:
- In 4 stages i.e. with 4 FFs, the register can store upto 4 bits of data.
- Serial data is applied at the D input of the first FF. The Q output of the first FF is connected to the D input of the second FF, the output of the second FF is connected to the D input of the third FF and the Q output of the third FF is connected to the D input of the fourth FF. The data is outputted from the Q terminal of the last FF.
- When a serial data is transferred to a register, each new bit is clocked into the first FF at the positive going edge of each clock pulse.
- The bit that is previously stored by the first FF is transferred to the second FF.
- The bit that is stored by the second FF is transferred to the third FF, and so on.
- The bit that was stored by the last FF is shifted out.
- A shift register can also be constructed using J-K FFs or S-R FFs as shown in the figure below.

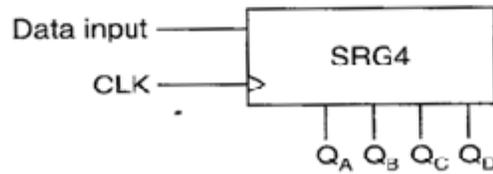


## SERIAL IN, PARALLEL OUT SHIFT REGISTER:-

- In this type of register, the data bits are entered into the register serially, but the data stored in the register is shifted out in the parallel form.
- When the data bits are stored once, each bit appears on its respective output line and all bits are available simultaneously, rather than bit – by – bit basis as in the serial output.
- The serial in, parallel out shift register can be used as a serial in, serial out shift register if the output is taken from the Q terminal of the last FF.
- The logic diagram and logic symbol of a 4 bit serial in, parallel out shift register is given below.



(a) Logic diagram

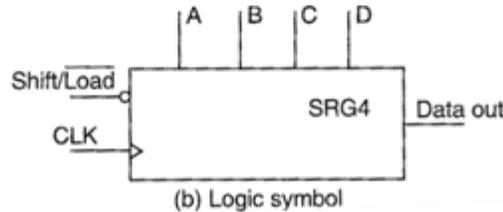
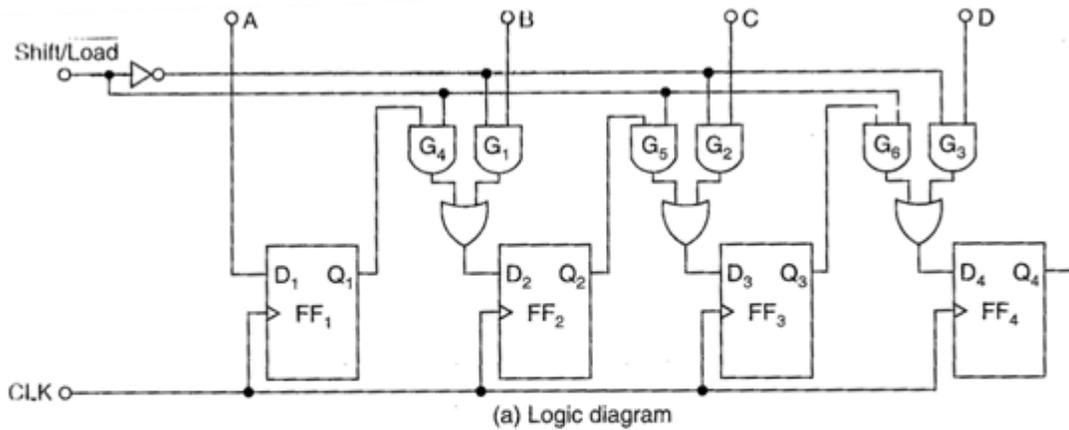


(b) Logic symbol

#### A 4-bit serial in, parallel out shift register

### PARALLEL IN, SERIAL OUT SHIFT REGISTER:-

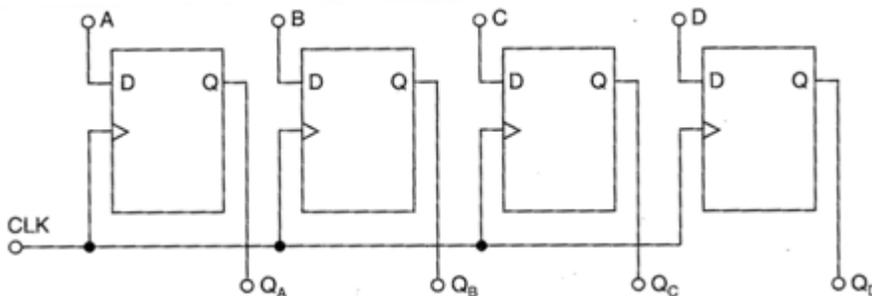
- For parallel in, serial out shift register the data bits are entered simultaneously into their respective stages on parallel lines, rather than on bit by bit basis on one line as with serial data inputs, but the data bits are transferred out of the register serially, i.e., on a bit by bit basis over a single line.
- The logic diagram and logic symbol of 4 bit parallel in, serial out shift register using D FFs is shown below.
- There are four data lines A, B, C and D through which the data is entered into the register in parallel form.
- The signal Shift  $\overline{\text{LOAD}}$  allows
  1. The data to be entered in parallel form into the register and
  2. The data to be shifted out serially from terminal  $Q_4$ .
- When Shift  $\overline{\text{LOAD}}$  line is HIGH, gates G1, G2, and G3 are disabled, but gates G4, G5 and G6 are enabled allowing the data bits to shift right from one stage to next.
- When Shift  $\overline{\text{LOAD}}$  line is LOW, gates G4, G5 and G6 are disabled, whereas gates G1, G2 and G3 are enabled allowing the data input to appear at the D inputs of the respective FFs.
- When clock pulse is applied, these data bits are shifted to the Q output terminals of the FFs and therefore the data is inputted in one step.
- The OR gate allows either the normal shifting operation or the parallel data entry depending on which AND gates are enabled by the level on the Shift  $\overline{\text{LOAD}}$  input.



**A 4- bit parallel in, serial out shift register**

**PARALLEL IN, PARALLEL OUT SHIFT REGISTER:-**

- In a parallel in, parallel out shift register, the data entered into the register in parallel form and also the data taken out of the register in parallel form. Immediately following the simultaneous entry of all data bits appear on the parallel outputs.
- The figure shown below is a 4 bit parallel in parallel out shift register using D FFs.
- Data applied to the D input terminals of the FFs.
- When a clock pulse is applied at the positive edge of that pulse, the D inputs are shifted into the Q outputs of the FFs.
- The register now stores the data.
- The stored data is available instantaneously for shifting out in parallel form.

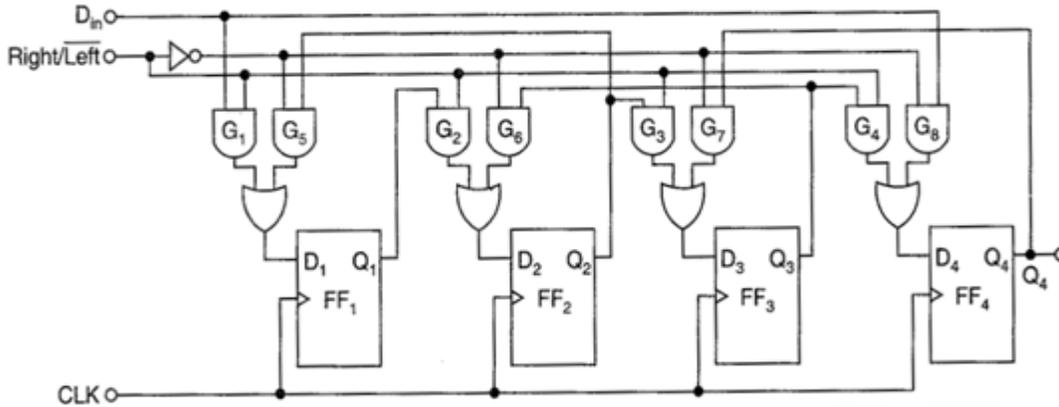


**Logic diagram of a 4 – bit parallel in, parallel out shift register**

**BIDIRECTIONAL SHIFT REGISTER:-**

- In bidirectional shift register is one in which the data bits can be shifted from left to right or from right to left.
- The figure shown below the logic diagram of a 4 bit serial in, serial out, bidirectional ( shift-left, shift-right) shift register.
- Right /Left is the mode signal. When Right /Left is a 1, the logic circuit works as a shift right shift register. When Right /Left is a 0, the logic circuit works as a shift right shift register.

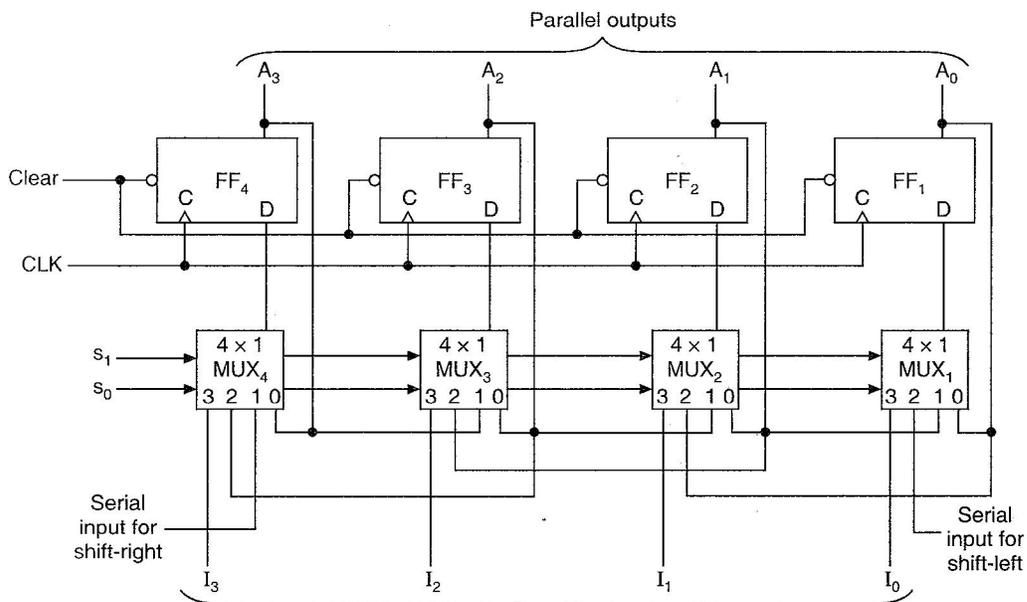
- The bidirectional is achieved by using the mode signal and two AND gates and one OR gate for each stage.
- A HIGH on the Right/Left control input enables the AND gates  $G_1, G_2, G_3$  and  $G_4$  and disables the AND gates  $G_5, G_6, G_7$  and  $G_8$  and the state of Q output of each FF is passed through the gate to the D input of the following FF. When clock pulse occurs, the data bits are effectively shifted one place to the right.
- A LOW Right/Left control input enables the AND gates  $G_5, G_6, G_7$  and  $G_8$  and disables the AND gates  $G_1, G_2, G_3$  and  $G_4$  and the Q output of each FF is passed to the D input of the preceding FF. When clock pulse occurs the data bits are then effectively shifted one place to the left.
- So, the circuit works as a bidirectional shift register.



**Logic diagram of 4- bit bidirectional shift register**

## UNIVERSAL SHIFT REGISTERS:-

- The register which has both shifts and parallel load capabilities, it is referred as a universal shift register. So, universal shift register is a bidirectional register, whose input can be either in serial form or in parallel form and whose output also can be either in serial form or parallel form.
- The universal shift register can be realized using multiplexers.
- The figure shows the logic diagram of a 4 bit universal shift register that has all the capabilities of a general shift register.



**Fig- (a) 4 bit universal shift register**

- It consists of four D flip-flops and four multiplexers.
- The four multiplexers have two common selection inputs  $S_1$  and  $S_0$ .
- Input 0 in each multiplexer is selected when  $S_1S_0 = 00$ , input 1 is selected when  $S_1S_0 = 01$ , and input 2 is selected when  $S_1S_0 = 10$  and input 3 is selected when  $S_1S_0 = 11$ .
- The selection inputs control the mode of operation of the register is according to the function entries shown in the table.
- When  $S_1S_0 = 00$  the present value of the register is applied to the D inputs of flip-flops. This condition forms a path from the output of each FF into the input of the same FF.
- The next clock edge transfers into each FF the binary value it held previously, and no change of state occurs.
- When  $S_1S_0 = 01$ , terminal 1 of the multiplexer inputs have a path of the D inputs of the flip-flops. This causes a shift right operation, with serial input transferred into FF<sub>4</sub>.
- When  $S_1S_0 = 10$  a shift left operation results with the other serial input going into the FF<sub>1</sub>.
- Finally when  $S_1S_0 = 11$ , the binary information on the parallel input lines is transferred into the register simultaneously during the next clock edge.

**Functional table for the register of fig – a:**

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

## **APPLICATIONS OF SHIFT REGISTERS:-**

### **1. Time delays:**

- In digital systems, it is necessary to delay the transfer of data until the operation of the other data have been completed, or to synchronize the arrival of data at a subsystem where it is processed with other data.
- A shift register can be used to delay the arrival of serial data by a specific number of clock pulses, since the number of stages corresponds to the number of clock pulses required to shift each bit completely through the register.
- The total time delay can be controlled by adjusting the clock frequency and by the number of stages in the register.
- In practice, the clock frequency is fixed and the total delay can be adjusted only by controlling the number of stages through which the data is passed.

### **2. Serial / Parallel data conversion:**

- Transfer of data in parallel form is much faster than that in serial form.
- Similarly the processing of data is much faster when all the data bits are available simultaneously. Thus in digital systems in which speed is important so to operate on data parallel form is used.
- When large data is to be transmitted over long distances, transmitting data on parallel lines is costly and impracticable.
- It is convenient and economical to transmit data in serial form, since serial data transmission requires only one line.

- Shift registers are used for converting serial data to parallel form, so that a serial input can be processed by a parallel system and for converting parallel data to serial form, so that parallel data can be transmitted serially.
- A serial in, parallel out shift register can be used to perform serial-to parallel conversion, and a parallel in, serial out shift register can be used to perform parallel- to –serial conversion.
- A universal shift register can be used to perform both the serial- to – parallel and parallel-to-serial data conversion.
- A bidirectional shift register can be used to reverse the order of data.

### **RING AND JOHNSON COUNTER:-**

- Ring counters are constructed by modifying the serial-in, serial-out, shift register.
- There are two types of ring counters
  - i) Basic ring counter
  - ii) Johnson counter
- The basic ring counter can be obtained from a serial-in serial- out shift register by connecting the Q output of the last FF to the D input of the first FF.
- The Johnson counter can be obtained from serial-in, serial- out, shift register by connecting the Q output of the last FF to the D input of the first FF.
- Ring counter outputs can be used as a sequence of synchronizing pulses.
- The ring counter is a decimal counter.

# D/A and A/D Converter

## Weighted Register Network

The most significant bit (MSB) resistance is one-eighth of the least significant bit (LSB) resistance.  $R_L$  is much larger than  $8R$ . The voltages  $V_A$ ,  $V_B$ ,  $V_C$  and  $V_D$  can be either equal to  $V$  (for logic 1) or 0 (for logical 0). Thus there are  $2^4 = 16$  input combinations from 0000 to 1111. The output voltage  $V_0$ , given by Millman's theorem is

$$V_0 =$$

When input is 0001,  $V_A = V_B = V_C = 0$  and  $V_D = V$  and output is  $V/15$ . If input is 0010,  $V_A = V_B = V_D = 0$  and  $V_C = V$  giving an output of  $2V/15$ . If input is 0011,  $V_A = V_B = 0$  and  $V_C = V_D = V$  giving an output of  $3V/15$ . Thus, the output voltage varies from 0 to  $V$  in steps of  $V/15$ .

## Binary Ladder Network

The weighted resistor network requires a range of resistor values. The binary ladder network requires only two resistance values. From node 1, the resistance to the digital source is  $2R$  and resistance to ground is also  $2R$ . From node 2, the resistance to digital source is  $2R$  and resistance to ground =

$$R + (2R)(2R) / (2R+2R) = 2R$$

Thus, from each of the nodes 1,2,3,4, the resistance to source and ground is  $2R$  each. A digital input 0001 means that D is connected to  $V$  and A, B, C are grounded. The output voltage  $V_0$  is  $V/16$ . Thus as input varies from 0000 to 1111, the output varies from  $V/16$  to  $V$  in steps of  $V/16$ .

A complete digital-to-analog converter circuit consists of a number of ladder networks (to deal with more bits of data), operational amplifier, gates etc.

## Performance Characteristics of D/A converters

The performance characteristics of D/A converters are resolution, accuracy, linear errors, monotonicity, setting time and temperature sensitivity.

- (a) **Resolution:** It is the reciprocal of the number of discrete steps in the D/A output. Evidently resolution depends on the number of bits. The percentage resolution is  $[1 / (2^N - 1)] * 100$  where  $N$  is the number of bits. The percentage resolution for different values of  $N$  is given in table.
- (b) **Accuracy:** It is a measure of the difference between actual output and expected output. It is expressed as a percentage of the maximum output voltage. If the maximum output voltage (or full scale deflection) is  $5\text{ V}$  and accuracy is  $\pm 0.1\%$ , then the maximum error is  $\frac{0.1}{100} * 5 = 0.005\text{ V}$  or  $5\text{ mV}$ . Ideally the accuracy should be better than  $\pm 0.5$  of LSB. In an 8 bit converter, LSB is  $1/256$  or  $0.39\%$  of full scale. The accuracy should be better than  $0.2\%$ .
- (c) **Setting Time:** When the input signal changes, it is desirable that analog output signal should immediately show the new output value. However in actual practice, the D/A converter takes some time to settle at the new position of the output voltage. Setting time is defined as the time taken by the D/A converter to settle with  $\pm 1/2$  LSB of its final value when a change in input digital signal occurs. The final time taken to settle down to new value is due to the transients and oscillations in the output voltage. Figure shows the definition of setting time.

| 3 bit Binary word | Analog voltage |
|-------------------|----------------|
| 000               | 0              |
| 001               | 1              |
| 010               | 2              |
| 011               | 3              |
| 100               | 4              |
| 101               | 5              |
| 110               | 6              |
| 111               | 7              |

Fig 1

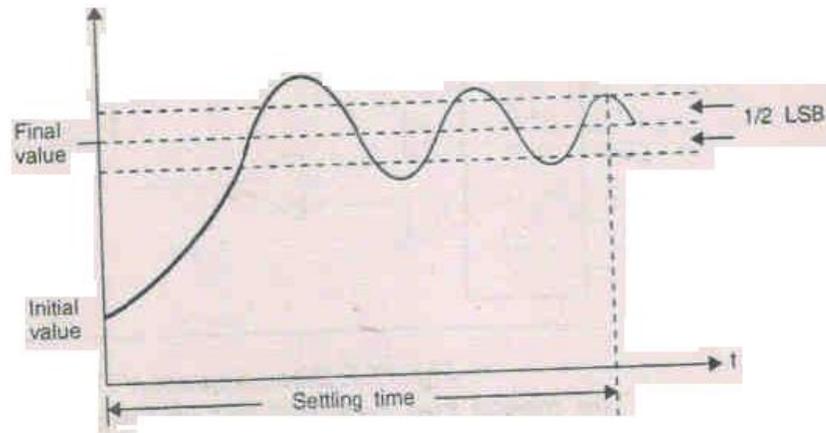


Fig. Settling time of D/A converter

### Quantization error:

An analog to digital converter changes analog signal into digital signal. It is important to note that in D/A converter the number of input is fixed. In 4 bit D/a converter there are 16 possible inputs and in 6 bit D/A converter there are 64 possible inputs. However, in A/D converter the analog input voltage can have any value in the specified range but the digital output can have only  $2^N$  discrete levels (for N bit converter). This means that there is a certain range of input voltage which correspond to every discrete output level.

Consider a 4 bit A/D converter having a resolution of 1 count per 100 mV. Fig (b) shows the analog input and digital output. It is seen that for input voltage range of 50 mV to 150 mV, the output is same i.e. 0001, for input voltage range of 150 mV to 250 mV, the output is the same, i.e. 0010. Thus we have one digital output for each 100 mV input range. If the digital signal of 0010 is fed to a D/A converter, it will show an output of 200 V whereas the original input voltage was between 150 V and 250 v. This error is called quntisation error and in this case this quntisation error can be  $\pm 50$  mV and is equal to  $\pm 1/2$  LSB.

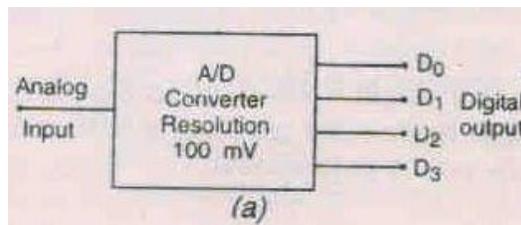


Fig (a) A/D Converter

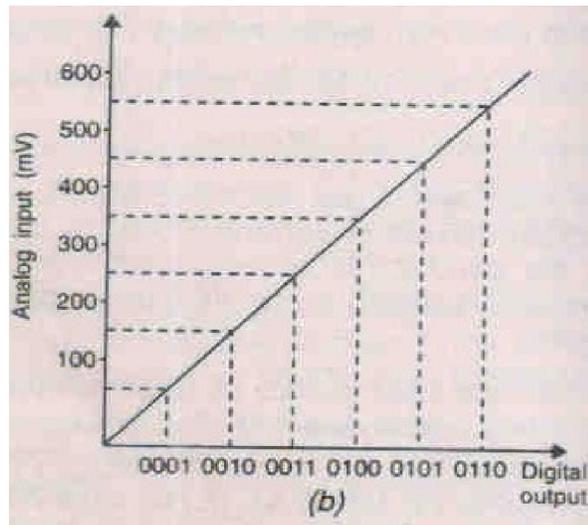


Fig (b) Quantisation error

**Stair Step A/D Converter / Ramp A/D converter:**

This converter is also called digital ramp or the counter type A/D converter. Figure shows the configuration for 8 bit converter. As seen in figure it uses a D/A converter and a binary counter to produce the digital number corresponding to analog input. The main components are comparator, AND gate, D/A converter, divide by 256 counter and latches. The analog input is given to non-inverting terminal of comparator. The D/A converter provides stair step reference voltage.

Let the counter be in reset state and output of D/A converter be zero. An analog input is given to non-inverting terminal of comparator. Since the reference input is 0, the comparator gives High output and enables the AND gate. The clock pulses cause advancing of counter through its binary states and stair step reference voltage is produced from D/A converter. As the counter keeps advancing, successively higher stair step output voltage is produced. When this stair step voltage reaches the level of analog input voltage, the comparator output goes Low and disables the AND gate. The clock pulses are cut off and counter stops. The state of counter at this point is equal to the number of steps in reference voltage at which comparison occurs. The binary number corresponding to this number of steps is the value of the analog input voltage. The control logic causes this binary number to be loaded into the latches and counter is reset.

This converter is rather slow in action because the counter has to pass through the maximum number of states before a conversion takes place. For 8 bit device this means 256 counter states.

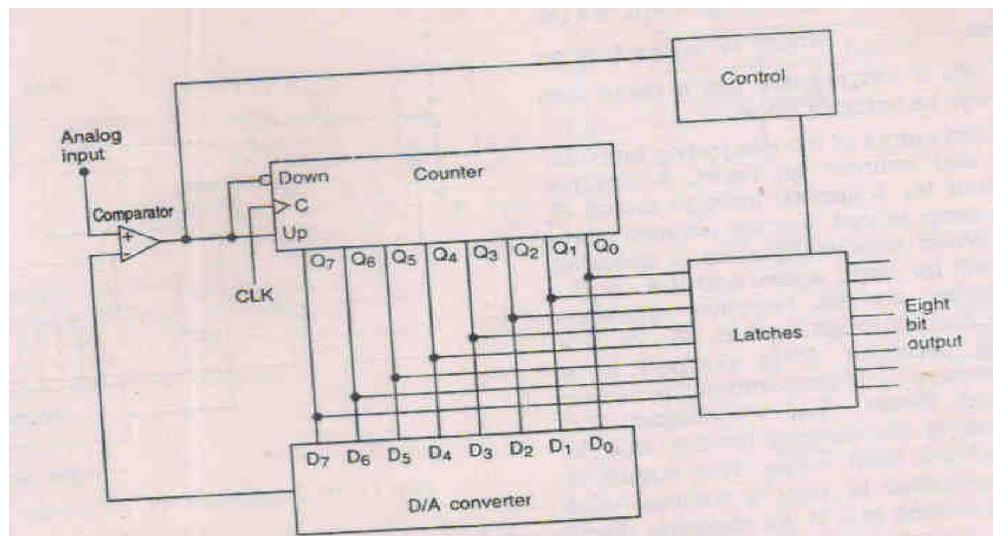


Fig (a) 8 bit up-down counter type A/D converter

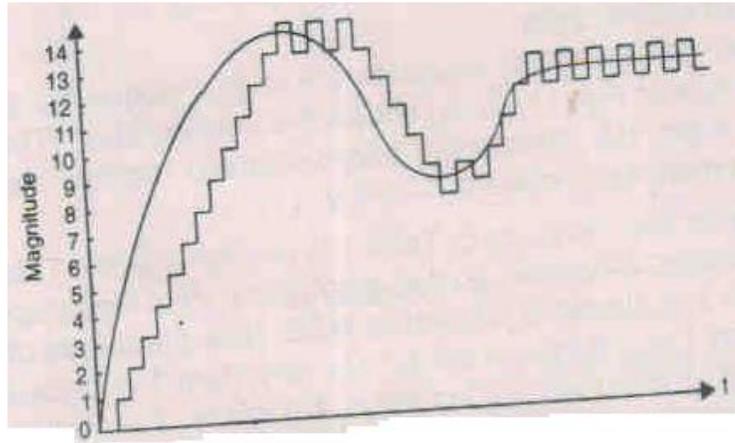


Fig (b) Tracking action of updown counter type A/D Converter

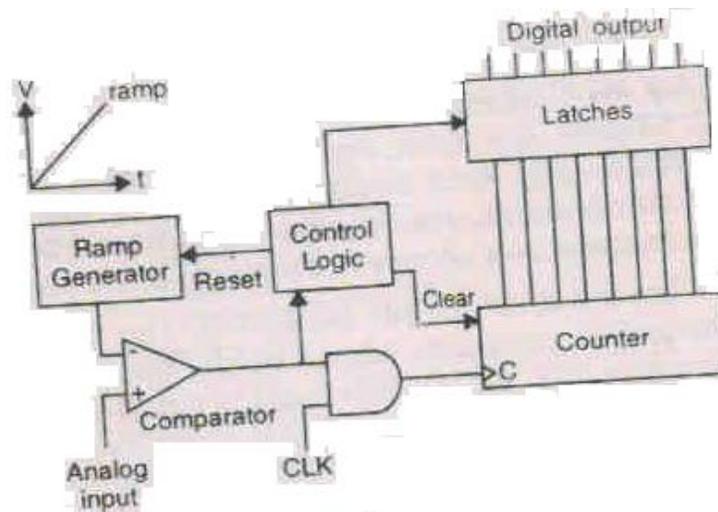


Fig (c) Single slope A/D converter

### Dual slope A/D converter:

The single slope A/D converter is susceptible to noise. The dual slope converter is free from this problem. It uses an op-amp used as integrating amplifier for ramp generator. It is dual slope device because it uses a fixed slope ramp as well as variable slope ramp. Fig. Shows the configuration.

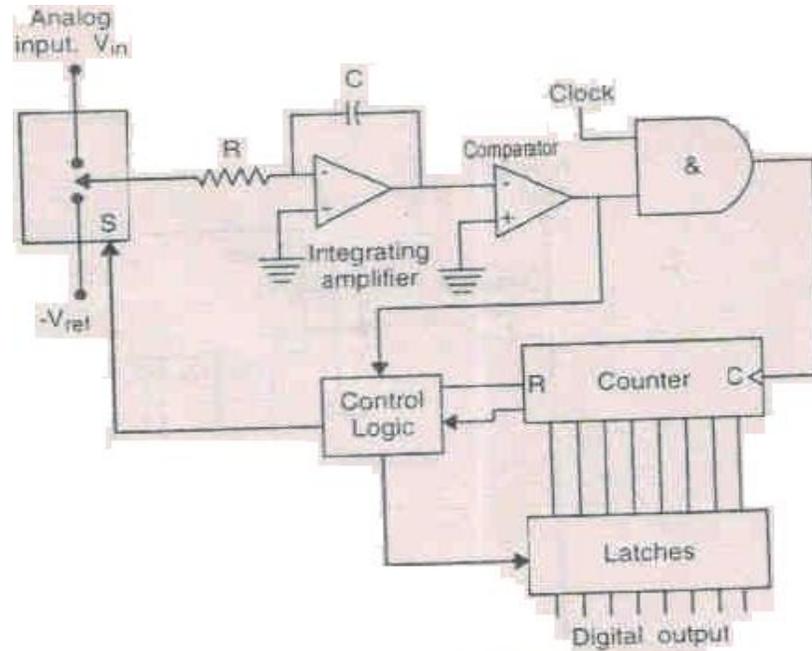
It is seen that the integrating op-amp uses a capacitor in the feedback path.

$$\text{Output voltage of integrating op-amp} = -\frac{1}{C} \int i dt = -\frac{1}{RC} \int V_{in} dt$$

Thus the output voltage is integral of analog input voltage. If  $V_{in}$  is constant, we get an output  $-V_{in} \frac{t}{RC}$  which is a fixed slope ramp. If  $V_{in}$  is varying we get a ramp with fixed as well as variable slope.

Let the output of the integrating amplifier be zero and counter be reset. A positive analog input  $V_{in}$  is applied through switch S, we get a ramp output and the counter starts working. When counter reaches a specified count, it will be reset again and the control logic switches on the negative reference voltage  $-V_{ref}$  (through switch S). At this instant the capacitor C is charged to a negative voltage  $-V$  proportional to analog input voltage. When  $-V_{ref}$  is connected the capacitor starts discharging linearly due to constant current from  $-V_{ref}$ . The output of integrating amplifier is now a positive fixed slope ramp starting at  $-V$ . As capacitor discharges, the counter advances from the reset state. When the output of integrator becomes zero, the comparator output

becomes Low and disables the clock signal to the AND gate. The counter is therefore stopped and the binary counter is latched. This completes one conversion cycle. The binary count is proportional to analog input  $V_{in}$ .



*Fig. Dual slope A/D converter*

### Successive Approximation A/D Converter:

This is the most widely used A/D converter. As the name suggests the digital output tends towards analog input through successive approximations. Fig. Shows the configuration. The main components are op-amp comparator, control logic, SA (successive approximation) register and D/A converter. As shown it is a six bit device using a maximum reference of 64 V.

Let the analog input be 26.1 v. The SA register is first set to zero. Then 1 is placed in MSB. This is fed to D/A converter whose output goes to comparator. Since the analog input (26.1 V) is less than D/A output (i.e. 32 V) the MSB is set to zero. Then 1 is placed in bit next to MSB. Now the output of D/A is 16 V. Since analog input is more than 16 V, this 1 is retained in this bit position. Next 1 is placed in third bit position. Now the D/A output is 24 V which is less than analog input. Therefore this 1 bit is retained and 1 is placed in the next bit. Now the D/A output is 28 V, which is more than analog input. Therefore this 1 bit is set to zero and 1 is placed in 5<sup>th</sup> bit position producing a D/A output of 26 V. It is less than analog input. Therefore this 1 bit is retained. Now 1 is placed in LSB producing a D/A output of 27 V which is more than analog input. Therefore LSB is set to zero and the converter gives an output of 26 V.

The successive approximation method of A/D converter is very fast and takes only about 250 ns/ bit.

### Performance Characteristics of A/D converters:

The performance characteristics of A/D converters are resolution, accuracy, A/D gain and drift and A/D speed.

- Resolution:** A/D resolution is the change in voltage input necessary for a one bit change in output. It can also be expressed as percent.
- A/D Accuracy:** The accuracy of A/D conversion is limited by the  $\pm 1/2$  LSB due to quantisation error and the other errors of the system. It is defined as the maximum deviation of digital output from the ideal linear reference line. Ideally it approaches  $\pm 1/2$  LSB.
- A/D gain and Drift:** A/D gain is the voltage output is divided by the voltage input at the linearity reference line. It can usually be zeroed out.

Drift means change in circuit parameters with time. Drift errors of upto  $\pm 1/2$  LSB will cause a maximum errors of one LSB between the first and the last transition. Very low drift is quite difficult to achieve and increases cost of the device.

(d) A/D speed: It can be defined in two ways, i.e. either the time necessary to do one conversion or the time between successive conversions at the highest rate possible. Speed depends on the settling time of components and the speed of the logic.

