
DIGITAL ELECTROICS
3rd Semester

SYLLABUS

Unit-1: Basics of Digital Electronics

Number System-Binary, Octal, Decimal, Hexadecimal - Conversion from one system to another number system.

Arithmetic Operation-Addition, Subtraction, Multiplication, Division, 1's &2's complement of Binary numbers& Subtraction using complements method

Digital Code & its application & distinguish between weighted & non-weight Code, Binary codes, excess-3 and Gray codes.

Logic gates: AND,OR,NOT,NAND,NOR, Exclusive-OR, Exclusive-NOR--Symbol, Function, expression, truth table & timing diagram

Universal Gates& its Realisation

Boolean algebra, Boolean expressions, Demorgan's Theorems.

Represent Logic Expression: SOP & POS forms

Karnaugh map (3 & 4 Variables)&Minimization of logical expressions ,don't care conditions

Unit-2: Combinational logic circuits

Half adder, Full adder, Half Subtractor, Full Subtractor, Serial and Parallel Binary 4 bit adder.
Multiplexer (4:1), De- multiplexer (1:4), Decoder, Encoder, Digital comparator(3 Bit)
Seven segment Decoder
(Definition, relevance, gate level of circuit Logic circuit, truth table, Applications of above)

Unit-3: Sequential logic Circuits

Principle of flip-flops operation, its Types,

SR Flip Flop using NAND,NOR Latch (un clocked)

Clocked SR,D,JK,T,JK Master Slave flip-flops-Symbol, logic Circuit, truth table and applications

Concept of Racing and how it can be avoided.

PREPARED BY **-BHABODEEPIKA MOHANTY**

Unit-4: Registers, Memories & PLD

Shift Registers-Serial in Serial -out, Serial- in Parallel-out, Parallel in serial out and Parallel in parallel out

Universal shift registers-Applications.

Types of Counter & applications

Binary counter, Asynchronous ripple counter (UP & DOWN), Decade counter. Synchronous counter, Ring Counter.

Concept of memories-RAM, ROM, static RAM, dynamic RAM,PS RAM

Basic concept of PLD & applications

Unit-5: A/D and D/A Converters

Necessity of A/D and D/A converters.

D/A conversion using weighted resistors methods.

D/A conversion using R-2R ladder (Weighted resistors)network.

A/D conversion using counter method.

A/D conversion using Successive approximate method

Unit-6: LOGIC FAMILIES

Various logic families &categories according to the IC fabrication process

Characteristics of Digital ICs- Propagation Delay, fan-out, fan-in, Power Dissipation ,Noise Margin ,Power Supply requirement &Speed with Reference to logic families.

Features, circuit operation &various applications of TTL(NAND), CMOS (NAND & NOR)

Unit-1: Basics of Digital Electronics

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 = [Integer part .Fractional part]

↑

Radix point

NUMBER SYSTEM:-

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

an an-1 an-2 …… a0 . a -1 a -2 a - m

This will be interpreted as

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

Where Y = value of the entire number

a_n = the value of the n^{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,

dn dn-1 dn-2 …… d0 . d -1 d -2 d - m

is given by

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

For example:-

$9256.26 = 9 \times 1000 + 2 \times 100 + 5 \times 10 + 6 \times 1 + 2 \times (1/10) + 6 \times (1/100)$

$= 9 \times 10^3 + 2 \times 10^2 + 5 \times 10^1 + 6 \times 10^0 + 2 \times 10^{-1} + 6 \times 10^{-2}$

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

$d_n \quad d_{n-1} \quad d_{n-2} \quad \ldots\ldots\ldots\ldots \quad d_0 \quad . \; d_{-1} \quad d_{-2} \quad d - k$

is given by

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

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 = 24 , every 4 – bit group of binary can be represented by an hexadecimal digit.

CONVERSION FROM ONE NUMBER SYSTEM TO ANOTHER :-

BINARY NUMBER SYSTEM:-

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:

Convert $(10101)_2$ to decimal.

Solution : (Positional weight) $2^4\ 2^3\ 2^2\ 2^1\ 2^0$

Binary number 10101

$= (1 \times 2^4) + (0 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0)$

$= 16 + 0 + 4 + 0 + 1$

$= (21)_{10}$

Convert $(111.101)_2$ to decimal.

Solution:

$(111.101)_2 = (1 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) + (1 \times 2^{-1}) + (0 \times 2^{-2}) + (1 \times 2^{-3})$

$= 4 + 2 + 1 + 0.5 + 0 + 0.125$

$= (7.625)_{10}$

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.

Octal	Binary	Octal	Binary
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 $(2571.34)_8$

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

Hexadecimal	Binary	Hexadecimal	Binary
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:

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 $(2DB)_{16}$

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

DECIMAL NUMBER SYSTEM:-

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:

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.

2	**52**	**— 0**
2	**26**	**— 0**
2	**13**	**— 0**
2	**6**	**— 1**
2	**3**	**— 0**
2	**1**	**— 1**
0		**— 1**

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

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

Integer part				Fraction part
2	105			0.15 x 2 = 0.30
2	52	—	1	0.30 x 2 = 0.60
2	26	—	0	0.60 x 2 = 1.20
2	13	—	0	0.20 x 2 = 0.40
2	6	—	1	0.40 x 2 = 0.80
2	3	—	0	0.80 x 2 = 1.60
2	1	—	1	
	0	—	1	

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

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	**378**			**0.93 x 8 = 7.44**
8	**47**	**—**	**2**	**0.44 x 8 = 3.52**
8	**5**	**—**	**7**	**0.52 x 8 = 4.16**
	0	**—**	**5**	**0.16 x 8 = 1.28**

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

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 2598			**0.675 x 16 = 10.8**	**A**
	16 162	**— 6**	**6**	**0.800 x 16 = 12.8**	**C**
	16 10	**— 2**	**2**	**0.800 x 16 = 12.8**	**C**
	0	**— 10**	**A**	**0.800 x 16 = 12.8**	**C**

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

OCTAL NUMBER SYSTEM:-

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 to binary	**= 011**	**110**	**111**	**.**	**101**	**010**

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

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:

$(4057.06)_8 = 4 \times 8^3 + 0 \times 8^2 + 5 \times 8^1 + 7 \times 8^0 + 0 \times 8^{-1} + 6 \times 8^{-2}$

$= 2048 + 0 + 40 + 7 + 0 + 0.0937$

$= (2095.0937)_{10}$

Result is $(2095.0937)_{10}$

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

HEXADECIMAL NUMBER SYSTEM :-

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 $(3A9E.B0D)_8$ is $(0011101010011110.101100001101)_2$

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 $(A0F9.0EB)_{16}$ to decimal

Solution:

$(A0F9.0EB)_{16} = (10 \times 16^3)+(0 \times 16^2)+(15 \times 16^1) +(9 \times 16^0) +(0 \times 16^{-1}) +(14 \times 16^{-2}) +(11 \times 16^{-3})$

$= 40960 + 0 + 240 + 9 + 0 +0.0546 + 0.0026$

$= (41209.0572)_{10}$

Result is $(41209.0572)_{10}$

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 $(B9F.AE)_{16}$ 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 $(5637.534)_8$

BINARY ARITHEMATIC OPERATION :-

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 $(100101)_2$ and $(1101111)_2$.

Solution :-

1 0 0 1 0 1

+ 1 1 0 1 1 1 1

1 0 0 1 0 1 0 0

Result is $(10010100)_2$

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 :-

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

Solution :- 1 0 1 0 . 0 1 0

- 1 1 1 . 1 1 1

0 0 1 0 .0 1 1

Result is $(0010.011)_2$

BINARY MULTIPLICATION:-

- The binary multiplication rules are as follows 0 x 0 = 0 ; 1 x 1 = 1 ; 1 x 0 = 0 ; 0 x 1 = 0

For example :-

Multiply $(1101)_2$ by $(110)_2$. Solution :-

1 1 0 1

x 1 1 0

0 0 0 0

1 1 0 1

+ 1 1 0 1

1 0 0 1 1 1 0

Result is $(1001110)_2$

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 :-

110) 101101 (111.1

\- 110

1010

110

1001

110

110

110

000

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 complement = 1's 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	**0**	**1**	**1**	**0**

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

SUBSTRACTION 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:-

1 1 0 1 0 1 1 0 1 0 = 26

- 1 0 0 0 0 => + 0 1 1 1 1 (1's complement) = - 16

Carry → 1 0 1 0 0 1 + 10

+ 1

0 1 0 1 0 = +10

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:-

	1 0 1 0 1 0 0		**1 0 1 0 1 0 0**	**=**	**84**
-	**1 0 1 0 1 0 0**	**=>**	**+ 0 1 0 1 1 0 0 (2's complement)**	**=**	**- 84_**
			1 0 0 0 0 0 0 0 (Ignore the carry)		**0**
		=	**0 (result = 0)**		

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 2n 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:-

$(567)_{10}$ 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_{10} (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 $(58.9)_{10}$

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:-

3 7		**0110 1010**	**(37 in XS-3)**
+ 2 8	**=>**	**+ 0101 1011**	**(28 in XS-3)**
6 5		**1011 11010**	**(Carry is generated)**
		+ 1	**(Propagate carry)**
		1100 0101	**(Add 0110 to correct 0101 and**
		- 0011 +0011	**subtract 0011 to correct 1100)**
		1001 1000	**(Corrected sum in XS-3 = 65_{10})**

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 27 = 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 group.

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} - - - - - B1 and its gray code equivalent by G_n G_{n-1} G1,

where B_n and G_n are the MSBs , then gray code bits are obtained from the binary code as follows $G_n = B_n$

$G_{n-1} = B_n B_{n-1}$

.

.

.

.

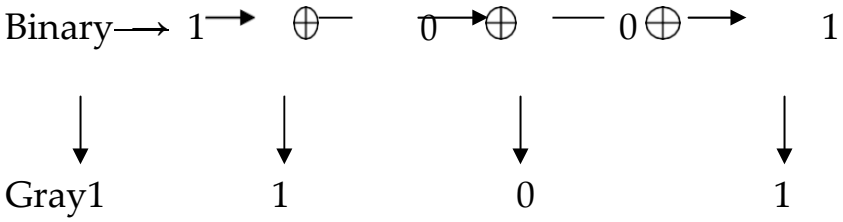
$G_1 = B_2 \oplus B_1$

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

For example :-

Convert the binary 1001 to the Gray code.

Solution :-`



The gray code is 1101

GRAY- TO - BINARY CONVERSION:-

- If an n-bit gray number is represented by $G_n\ G_{n-1}$ --------G_1 and its binary equivalent by $B_n\ B_{n-1}\ B1$, then binary bits are obtained from Gray bits as follows:
- $B_n = G_n$

$B_{n-1} = B_n \oplus G_{n-1}$

.

.

.

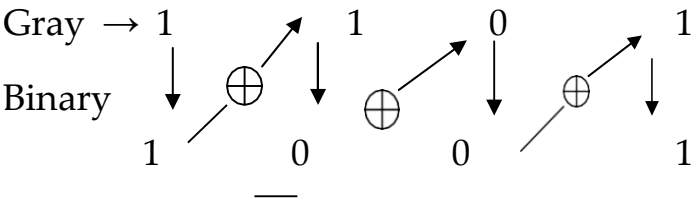
.

$B_1 = B_2 \oplus G_1$

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:-

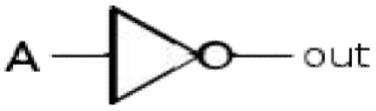
- A negative logic system is the one in which the lower of the two voltage levels represents the logic 1 and the higher of the two voltages level represents the logic 0.

DIFFERENT TYPES OF LOGIC GATES:- NOT GATE (INVERTER):-

- A NOT gate, also called and 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

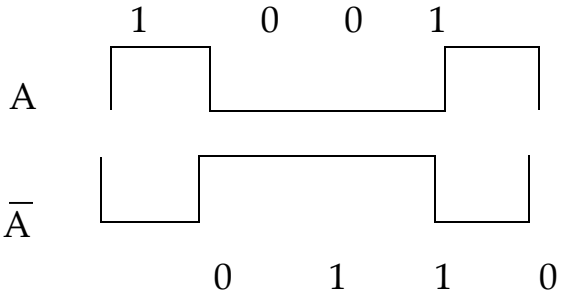
Logic Symbol



Truth table

INPUT A	OUTPUT 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.

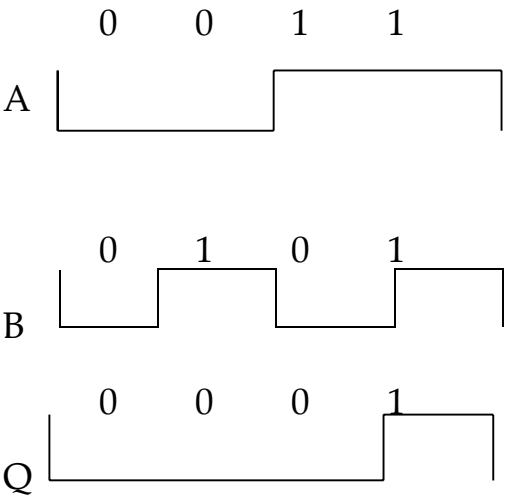
IC No.:- 7408

Logic Symbol

Truth Table

		OUTPUT
A	**B**	**Q=A . 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.

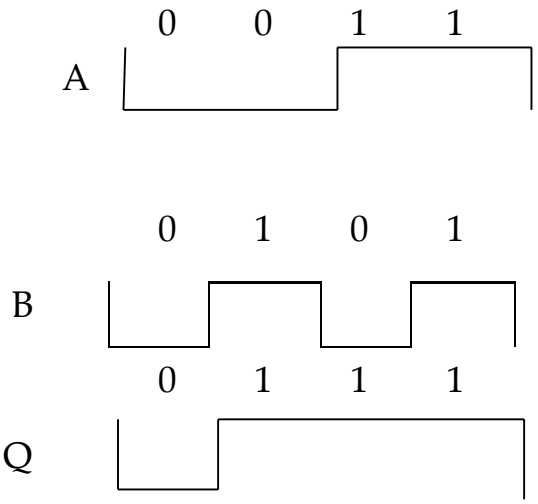
IC No.:- 7432

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

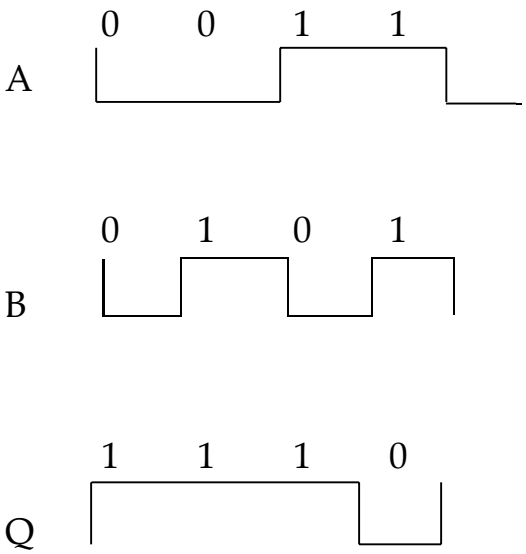
PREPARED BY **-BHABODEEPIKA MOHANTY**

Logic Symbol

Truth Table

INPUT		OUTPUT
A	**B**	**Q= A . 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.

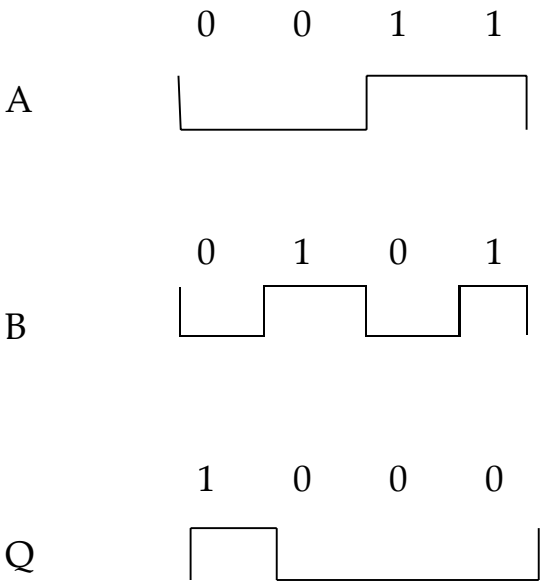
- IC No.:- 7402 two input NOR gate 7427 three input NOR gate 7425 four input NOR gate

Logic Symbol

Truth Table

INPUT		OUTPUT
A	**B**	**$\overline{Q}$= 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

Truth Table

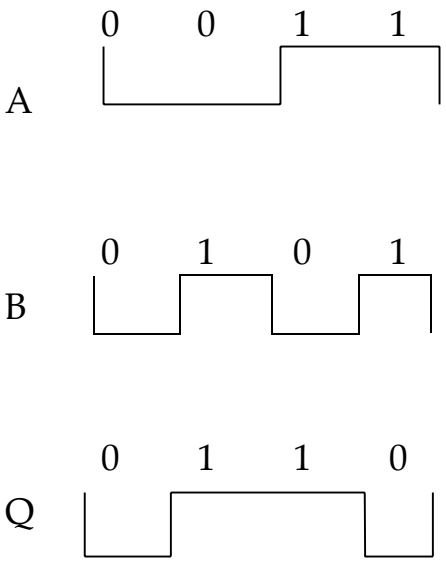
INPUT		OUTPUT
A	**B**	**Q = A⊕ B**
0	0	0
0	1	1
1	0	1
1	1	0

INPUTS are A and B

OUTPUT is Q = A ⊕B

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

Timing Diagram

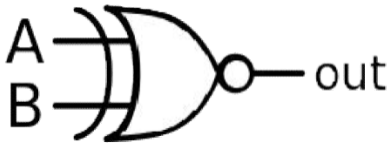


EXCLUSIVE – NOR (X-NOR) GATE:-

- 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

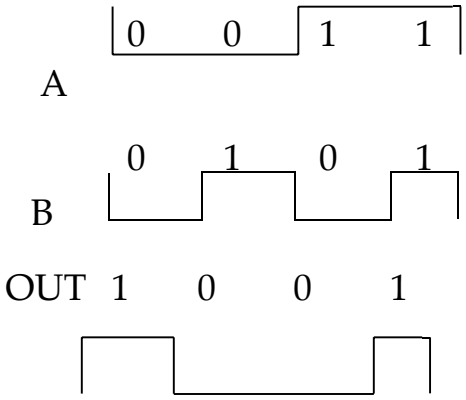


INPUT		OUTPUT
A	**B**	**OUT =A XNOR B**
0	0	1
0	1	0
1	0	0
1	1	1

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

$= A\ XNOR\ B$

Timing Diagram



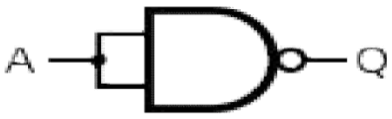
PREPARED BY **-BHABODEEPIKA MOHANTY**

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:-

Inverter from NAND gate



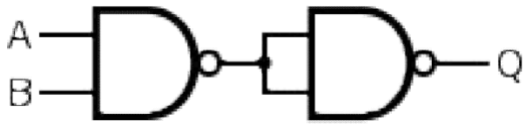
Input = A

Output $\overline{Q}$ = A

AND gate from NAND gate

Input s are A and B

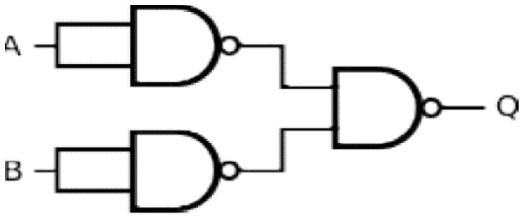
Output Q = A.B



OR gate from NAND gate

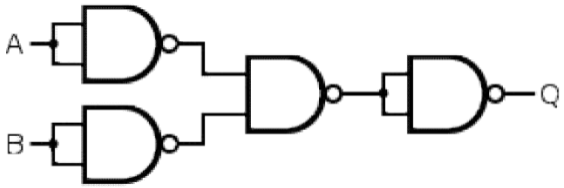
Inputs are A and B

Output Q = A+B



NOR gate from NAND gate

Inputs are A and B

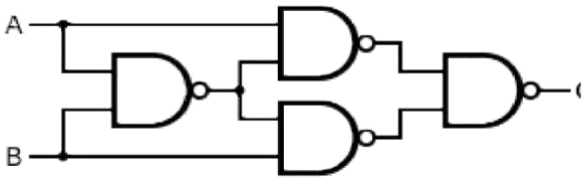


Output Q = A+B

EX-OR gate from NAND gate

Inputs are A and B

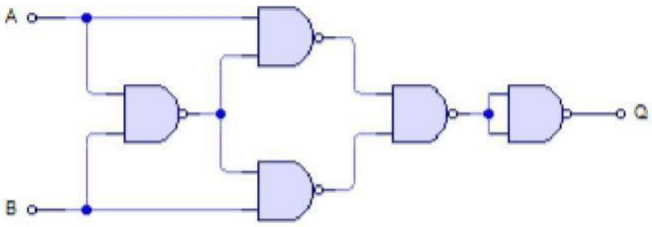
Output Q = A B + AB



EX-NOR gate From NAND gate

Inputs are A and B

Output Q = A B + A B



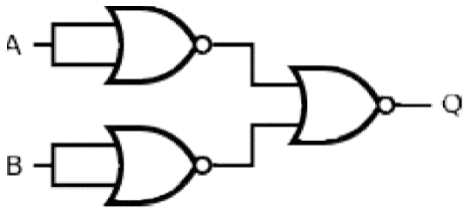
NOR GATE:-

Inverter from NOR gate

Input = A

Output Q = A

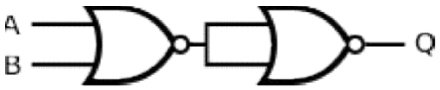
AND gate from NOR gate Input s are A and B Output Q = A.B



OR gate from NOR gate

Inputs are A and B

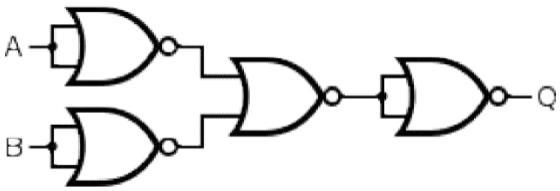
Output Q = A+B



NAND gate from NOR gate

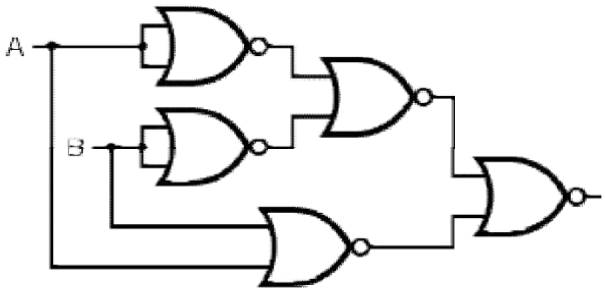
Inputs are $\overline{A}$ and B

Output Q = A.B



EX-OR gate from NOR gate

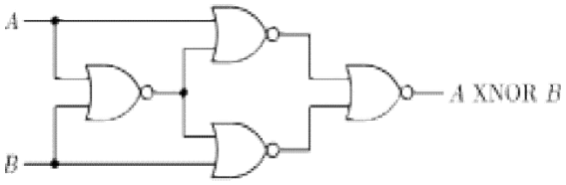
Inputs are A and B



Output Q = A B + AB

EX-NOR gate From NOR gate

Inputs are A and B



Output Q = A B + A B

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	OR operation	NOT operation
Axiom 1: 0 . 0 = 0	**Axiom 5: 0 + 0 = 0**	
		Axiom 9: 1 = 0
Axiom 2: 0 . 1 = 0	**Axiom 6: 0 + 1 = 1**	**Axiom 10:0 = 1**
Axiom 3: 1 . 0 = 0	**Axiom 7: 1 + 0 = 1**	
Axiom 2: 1 . 1 = 1	**Axiom 8: 1 + 1 = 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 $\overline{A} = 1$

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

Law 5:$\overline{\overline{A}} = 0$ (double complementation law)

OR Laws:-

The four OR laws are as follows :-

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

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

Law 3: A + A = A

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

AND Laws:-

The four AND laws are as follows:-

Law 1: A . 0 = 0(Null law)

Law 2: A . 1 = 1(Identity law)

Law 3: A . A = A

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

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 . B = B . 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.B. C = B. C. A = C. A. B = B. A. C

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	0	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 .B) C = A (B .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)

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)

ProofRHS = (A+B) (A+C)

= AA + AC + BA + BC

= A + AC + AB + BC

= A (1+ C + B) + BC

= A. 1 + BC (1 +C + B = 1 + B = 1)

= A + BC

= LHS

Redundant Literal Rule (RLR):-

Law 1: A + AB = A + B

Proof

A + AB = (A + A) (A + B)

= 1. (A + B)

= A +B

Law 2: A(A + B) = AB A(A + B) = AA + AB

= 0 + AB

= AB

8. Idempotence Laws:- Idempotence means same value.

Law 1: 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 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 · B = A

Proof

Proof

A + A · B = A (1 + B) = A · 1 = A

A	B	AB	A+AB
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

A	B	A+B	A(A+B)
0	0	0	0
0	1	1	0
1	0	1	1
1	1	1	1

Law 2: A (A + B) = A

Proof A (A + B) = A · A + A · B = A + AB = A(1 + B) = A · 1 = A

10. Consensus Theorem (Included Factor Theorem):-

Theorem 1:

AB +AC + BC = AB +AC

Proof

Theorem 2:

LHS = AB + AC + BC

= AB + AC + BC (A+A)

= AB + AC + BCA + BCA

= AB (1 + C) + AC (1+ B)

=AB (1) +AC (1)

= AB + AC

= RHS

Proof

(A + B)(A + C)(B + C) =(A +B)(A + C)

LHS = (A + B) (A + C) (B + C)

= (AA + AC + BA + BC) (B + C)

= (AC + BC +AB) (B + C)

= ABC + BC + AB + AC + BC+ABC

= AC + BC +AB

RHS= (A + B) (A+C)

= AA + AC + BC +AB

= AC + BC +AB

= LHS

11. Transposition Theorem:- Theorem:

AB + AC = (A + C)(A + B)

Proof

RHS= (A + C) (A + B)

= AA + CA + AB + CB

= 0 +AC + AB + BC
= AC + AB + BC (A+A)
= AB + ABC + AC +ABC
= AB + AC
= LHS

De Morgan's Theorem:-

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

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

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	$\bar{A}$	$\bar{B}$	$\bar{A}\,\bar{B}$
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: $\overline{A \cdot B} = \bar{A} + \bar{B}$

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

$[f(A, B, C, \ldots, 0, 1, +, \cdot)]_d = f(A, B, C, \ldots, 1, 0, \cdot, +)$

Relations between complement and dual

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

$$f_d(A, B, C, \ldots) = f(A, B, C, \ldots) = f_c(A, B, C, \ldots)$$

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

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

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.	$A \cdot A = A$	$A + \overline{A} = A$
8.	$A \cdot \overline{A} = 0$	$A + \overline{A} = 1$
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.	$\overline{A} \cdot (\overline{A} \cdot B) = A \cdot B$	$\overline{A + A} + \overline{B} = A + B$
14.	$\overline{AB} = \overline{A} + \overline{B}$	$\overline{A + B} = \overline{A}\,\overline{B}$
15.	$(A + B)(\overline{A} + C)(B + C) = (A + B)(\overline{A} + C)$	$AB + \overline{A}C + BC = AB + \overline{A}C$
16.	$A + BC = (A + B)(A + C)$	$A(B + C) = A\,B + A\,C$
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.	$\overline{A + B} = \overline{A}\,\overline{B} + \overline{A}B + A\overline{B}$	$\overline{A}\,\overline{B} = (A+B)(\overline{A}+B)(A+\overline{B})$
20.	$\overline{A}\,\overline{B} + \overline{A} + AB = 0$	$\overline{A + B} \cdot A \cdot (A + B) = 1$

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

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

- 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 mo, m1, m2 … .
- An 'n' variable function can have 2n 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) = ($\bar{A}$ + B + C·$\bar{C}$) + ($\bar{A}$ + $\bar{B}$ + C·C) –

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

- 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, …….
  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) = ∑m(0,2,4,6,7)

This has a complement that can be expressed as

$\overline{f}$ (A, B, C) = ∑ 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 =$(\overline{m_1 + m_3 + m_5})$ = $\bar{m}_1 . \bar{m}_3 . \bar{m}_5$

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

Example:-

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

Solution:-

In POS form

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

$A = A + B\,\overline{B} + C\overline{C}$

$= (A + B)\,(A + B) + C \cdot C$

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

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

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

Therefore

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

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

= (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:-

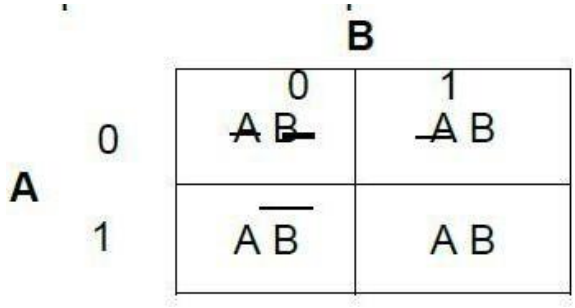
- The K- map is a chart or a graph, composed of an arrangement of adjacent cells, each representing a particular combination of variables in sum or product form.
- The K- map is systematic method of simplifying the Boolean expression.

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:-

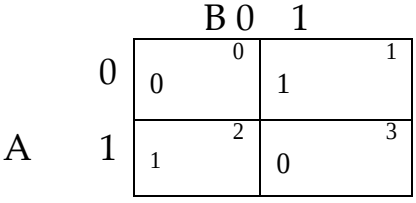
- The 2 variable K-map has $2^2 = 4$ squares. These squares are called cells.
- A '1' is placed in any square indicates that corresponding minterm is included in the output expression, and a 0 or no entry in any square indicates that the corresponding minterm does not appear in the expression for output.



Example:-

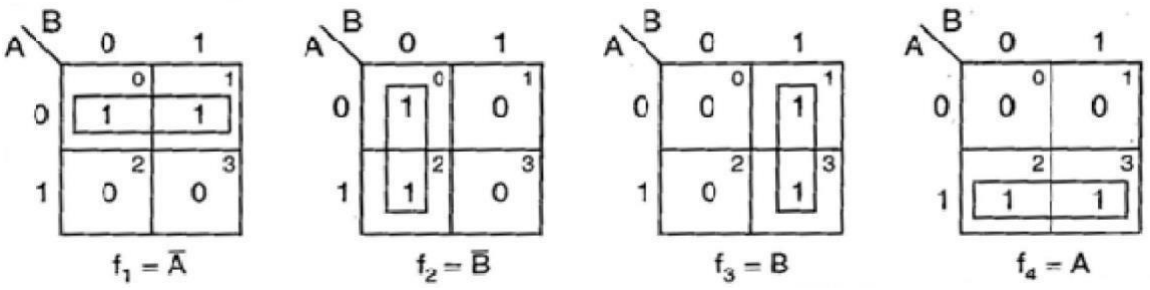
Map expression f= AB + AB Solution:-

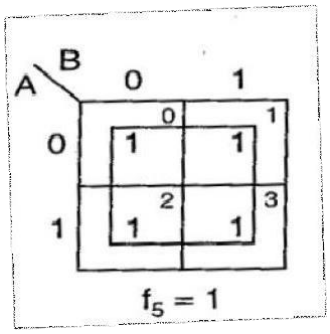
The expression minterms is $F = m_1 + m_2 = m(1, 2)$



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

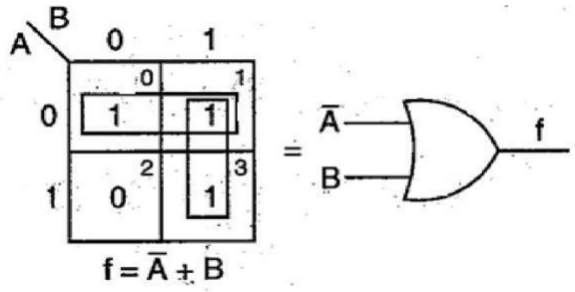
- Two minterms, which are adjacent to each other, can be combined to form a bigger square called 2 – square or a pair. This eliminates one variable that is not common to both the minterms.
- Two 2-squares adjacent to each other can be combined to form a 4-square. A 4- square eliminates 2 variables. A 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= $\bar{A}\bar{B} + \bar{A}B + AB$ using mapping.

Solution:-

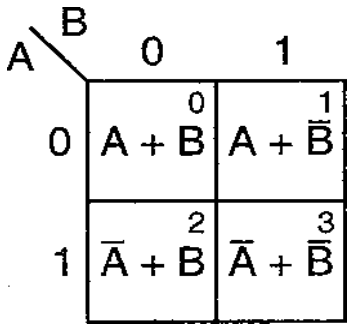
Expressed in terms of minterms, the given expression is $f = m_0 + m_1 + m_3 = \sum m(0, 1, 3)$



$F = \overline{A} + B$

Mapping of POS Expression:-

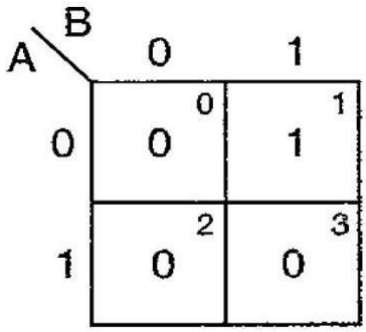
Each sum term in the standard POS expression is called a Maxterm. A function in two variables (A,B) has 4 possible maxterms, A + B, A + B, A + B and A + B . They are represented as M_0, M_1, M_2 and M_3 respectively.



The maxterm of a two variable K-map Example:-

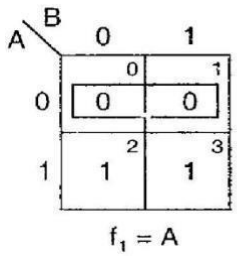
Plot the expression f= $(A + B)(\overline{A} + B)(\overline{A + B})$ Solution:-

Expression interms of maxterms is f = πM (0, 2, 3)

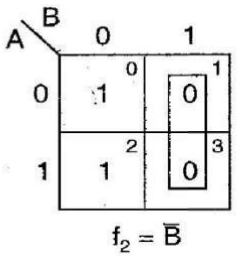


Minimization of POS Expressions:-

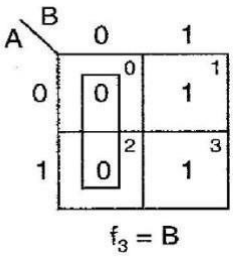
- In POS form the adjacent 0s are combined into large square as possible. If the squares having complemented variable then the value remain constant as a 1 and the non-complemented variable if its value remains constant as a 0 along the entire square and then their sum term is written.



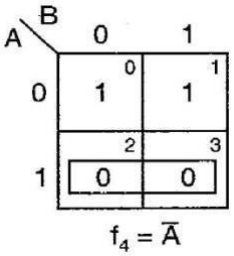
$f_1 = A$



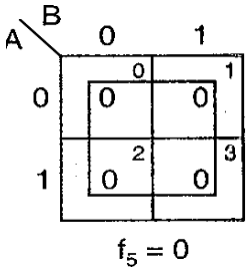
$f_2 = \overline{B}$



$f_3 = B$



$f_4 = \overline{A}$

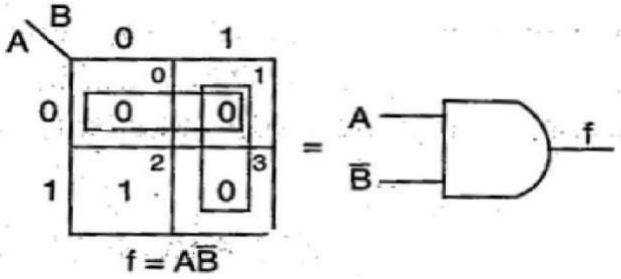


$f_5 = 0$

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

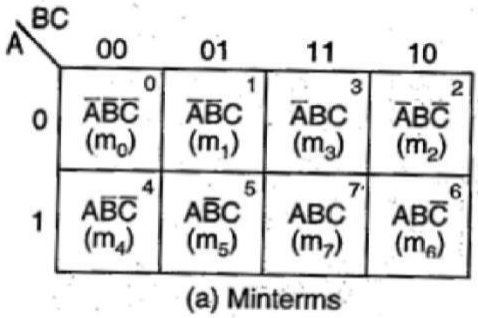
Reduce the expression f = $(A + \overline{B})(\overline{A} + \overline{B})(A + B)$ using mapping Solution:-

The given expression in terms of maxterms is f = πM (0, 1, 3)

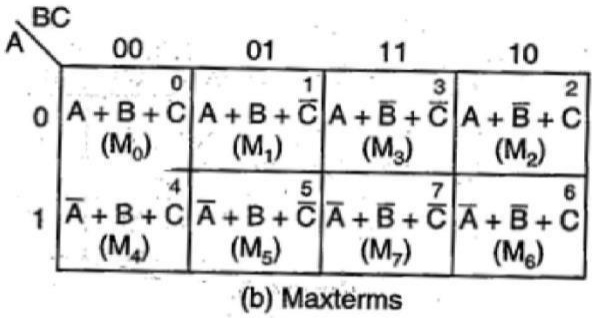


THREE VARIABLE K- MAP:-

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 square or cell and on the map represents each square minterm or maxterm is shown in figure below.



(a) Minterms

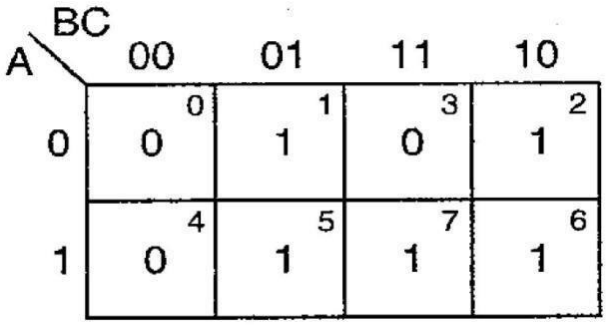


(b) Maxterms

Example:-

Map the expression f = ABC+ABC + ABC + ABC +ABC Solution:-

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

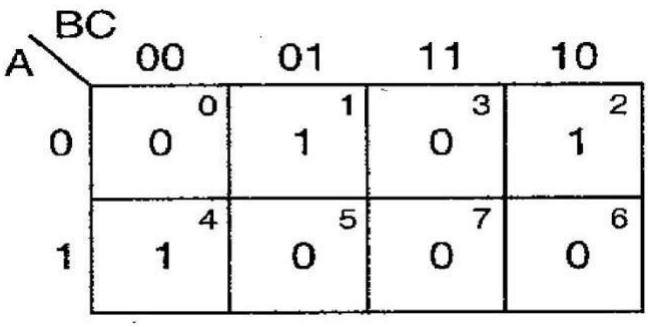


Example:-

– — — — – — — – —

Map the expression f = (A + B + C) (A + B+C) (A + B + C) (A + B + C) (A + B + C) Solution:-

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



Minimization of SOP and POS Expressions:-

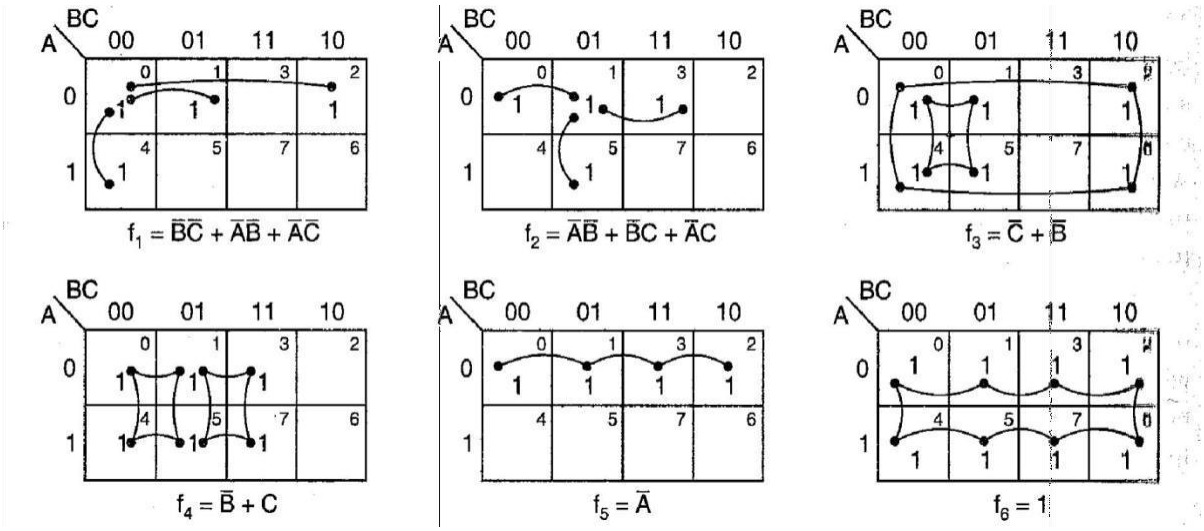
For reducing the Boolean expressions in SOP (POS) form the following steps are given below

- 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.

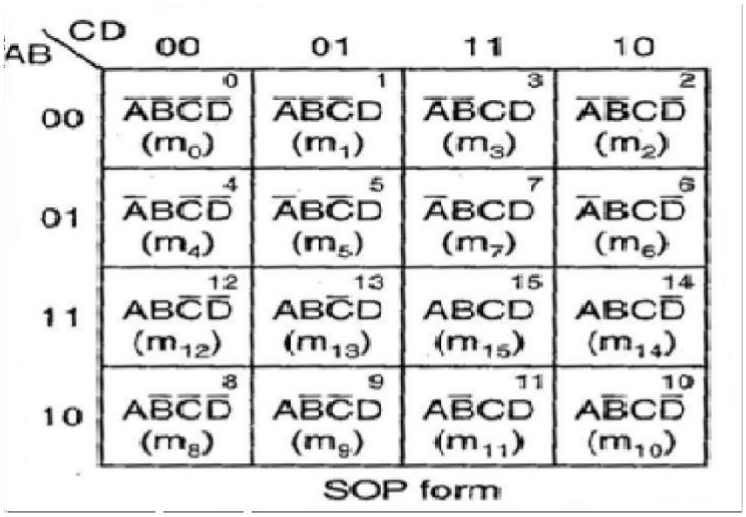
Some of the possible combinations of minterms in SOP form



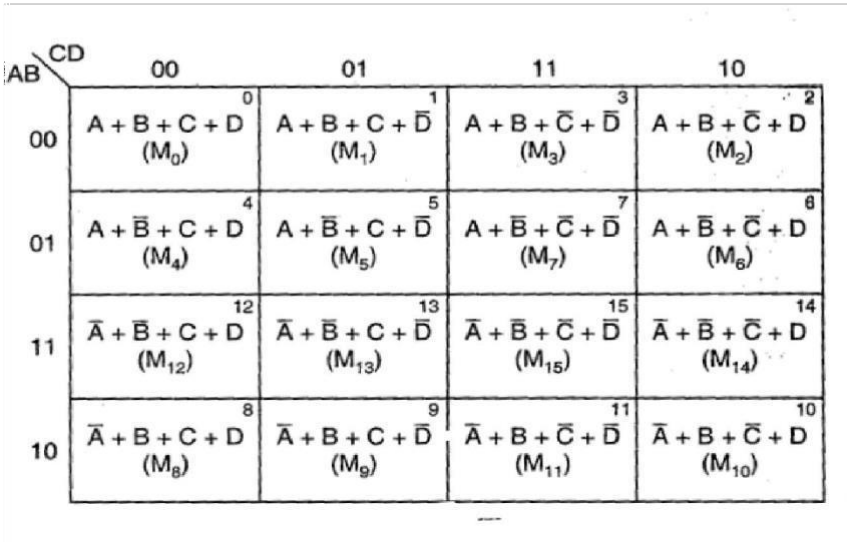
These possible combinations are also for POS but 1s are replaced by 0s.

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 desginations.



POS FORM



SOP FORM

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.

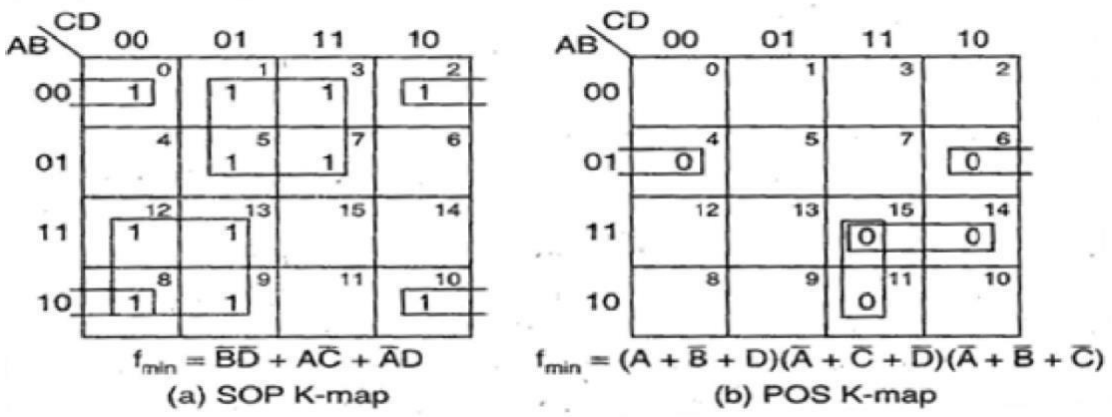
- For those 1s (0s) which are adjacent to only one other 1(0) make them pairs (2 squares). For quads (4- squares) and octet (8 squares) of adjacent 1s (0s) even if they contain some 1s (0s) which have already been combined. They must geometrically form a square or a rectangle.
- For any 1s (0s) that 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 = ∑ m (0, 1, 2, 3, 5, 7, 8, 9, 10, 12, 13)
Solution:-

The given expression in POS form is f = π M (4, 6, 11, 14, 15) and in SOP form f = ∑ m (0, 1, 2, 3, 5, 7, 8, 9,

10, 12, 13



The minimal SOP expression is $f_{min} = BD + AC + AD$

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

DON'T CARE COMBINATIONS:-

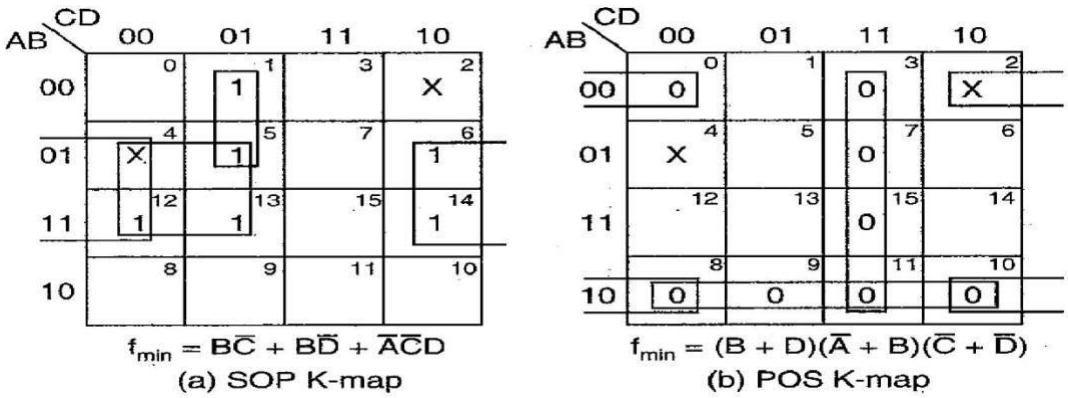
- The combinations for which the values of the expression are not specified are called don't care combinations or optional combinations and such expression stand incompletely specified. The output is a don't care for these invalid combinations. The don't care terms are denoted by d or X. During the process 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 = π M (0, 3, 7, 8, 9, 10, 11,15) + d(2, 4)



The minimal of SOP expression is f_{min} = BC + BD +ACD

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

Unit-2: Combinational logic circuits

- 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 1and logic 0.

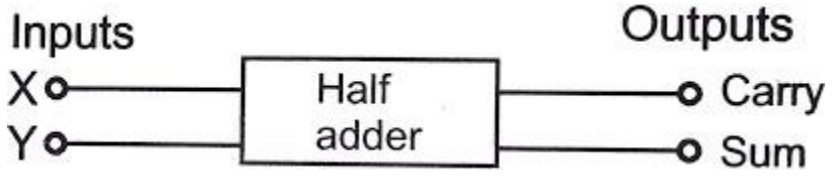
- **BINARY ADDER SUBTRACTOR:-**

    - 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.



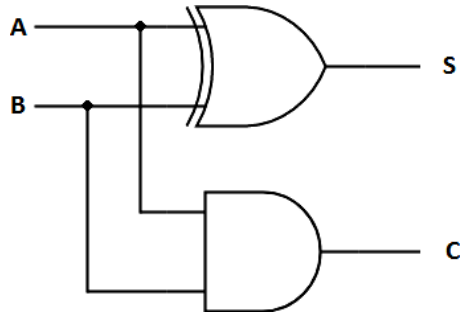
Truth Table

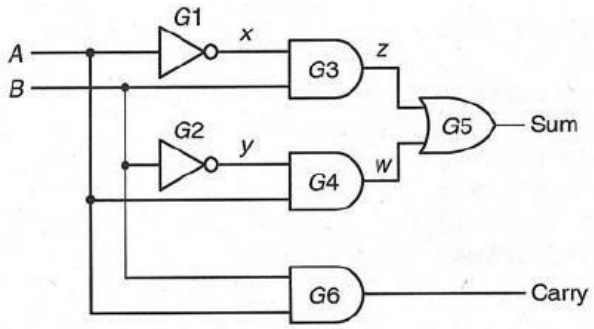
Input		Output	
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

- The simplified Boolean functions for the two outputs can be obtained directly from the 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.
- 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.





Logic diagram

For Carry

A \ B	0	1
0	0	0
1	0	(1)

Carry = AB

For Sum

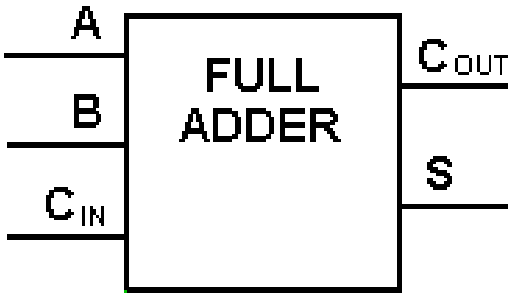
A \ B	0	1
0	0	(1)
1	(1)	0

Sum = $A\overline{B} + \overline{A}B$
= $A \oplus B$

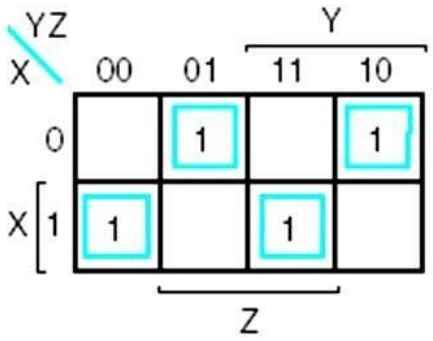
Maps for half-adder

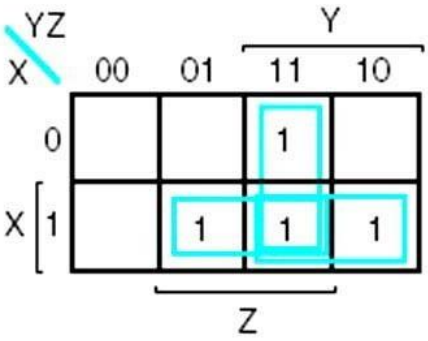
FULL ADDER:-

- 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.
  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.



Inputs			Outputs	
A	B	C_{in}	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1





$$S = \bar{X}\bar{Y}Z + \bar{X}Y\bar{Z} + X\bar{Y}\bar{Z} + XYZ$$
$$= X \oplus Y \oplus Z$$

$$C = XY + XZ + YZ$$
$$= XY + Z(X\bar{Y} + \bar{X}Y)$$
$$= XY + Z(X \oplus Y)$$

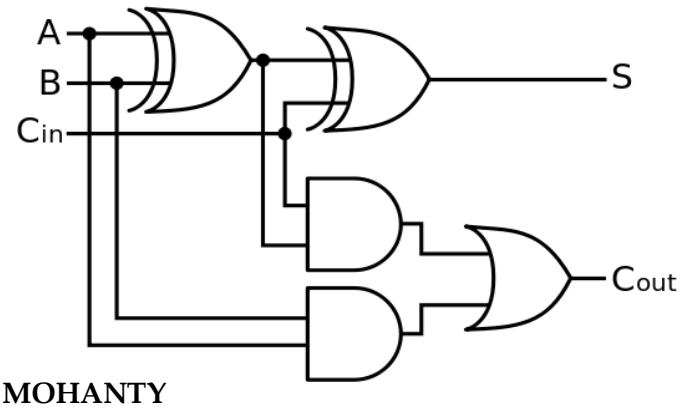
- The binary variable S gives the value of the least significant bit of the sum. The binary variable C gives the output carry formed by adding the input carry and the bits of the words.
- The eight rows under the input variables designate all possible combinations of the three variables. The output variables 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

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

$$C = xy + xz + y$$

- 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.

HALF SUBTRACTOR:-



- This circuit needs two

binary inputs and two binary outputs.

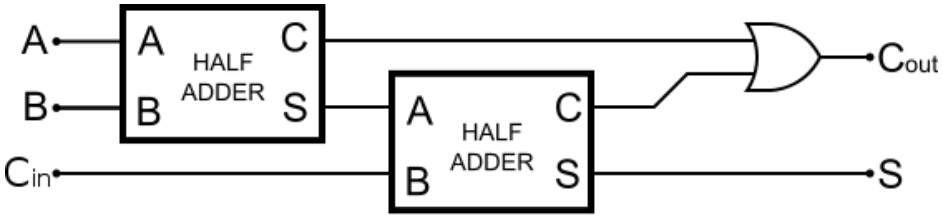
- Symbols x and y are assigned to the two inputs and D (for difference) and B (for borrow) to the outputs.
- The truth table for the half subtractor is listed in the below table.

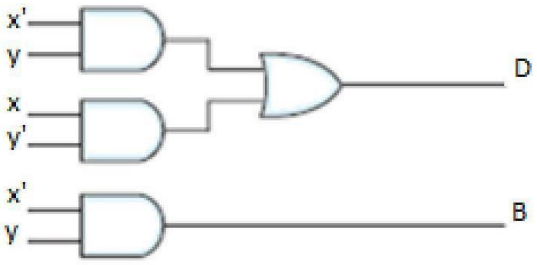
x	y	D	B
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Truth Table

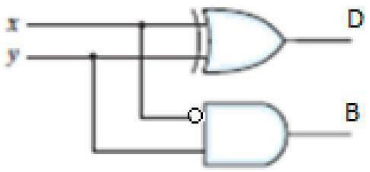
- The B output is 1 only when the inputs are 0 and 1. The D output represents the least significant bit of the subtraction.
- The subtraction operation is done by using the following rules as
    - 0-0=0;
    - 0-1=1 with borrow 1;
    - 1-0=1;
    - 1-1=0.
- The simplified Boolean functions for the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

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





D = x'y+xy'
B = x'y

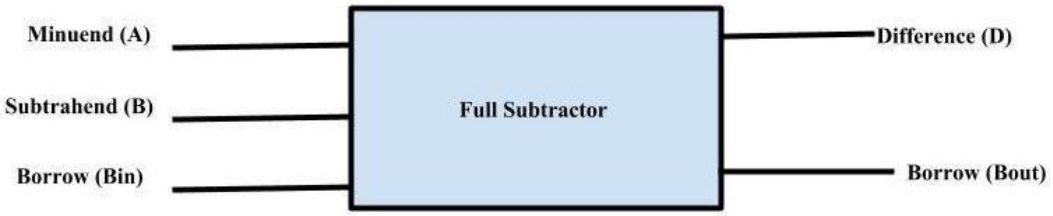


D = x ⊕ y
B=x'y

- 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 A and B , represent the two significant bits to be subtracted. The third input, B_{in} , is subtracted from the result 0f the first subtraction.



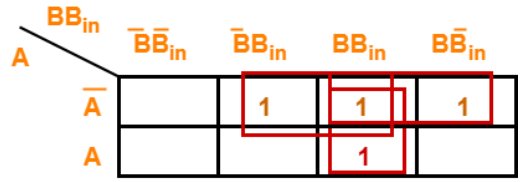
Input			Output	
A	B	C	Difference	Borrow
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

- 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.

For D:

$$D = A \oplus B \oplus B_{in}$$

For B_{in}:



$$B_{out} = \bar{A}B + (\bar{A} + B)B_{in}$$

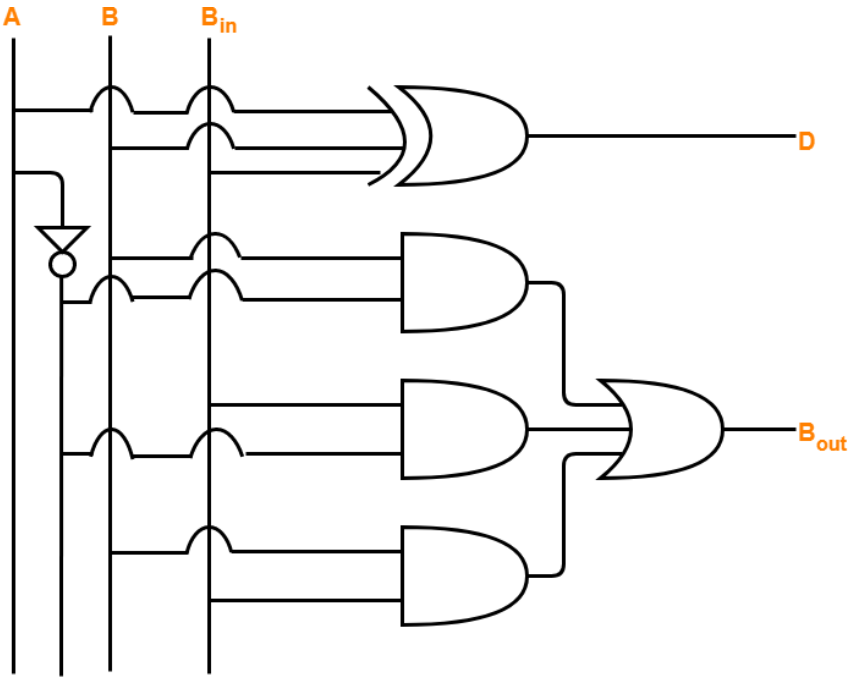
- The eight rows under the input variables designate all possible combinations of the three variables. The output variables are determined from the arithmetic subtraction of the input bits.
- The difference D becomes 1 when any one of the input is 1or all three inputs are equal to1 and the borrow B is 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

$$D = A'B'B_{in} + A'BB_{in}' + AB'B_{in}' + ABB_{in}$$

$$B = A'B_{in} + A'B + BB_{in}$$

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



Full Subtractor Logic Diagram

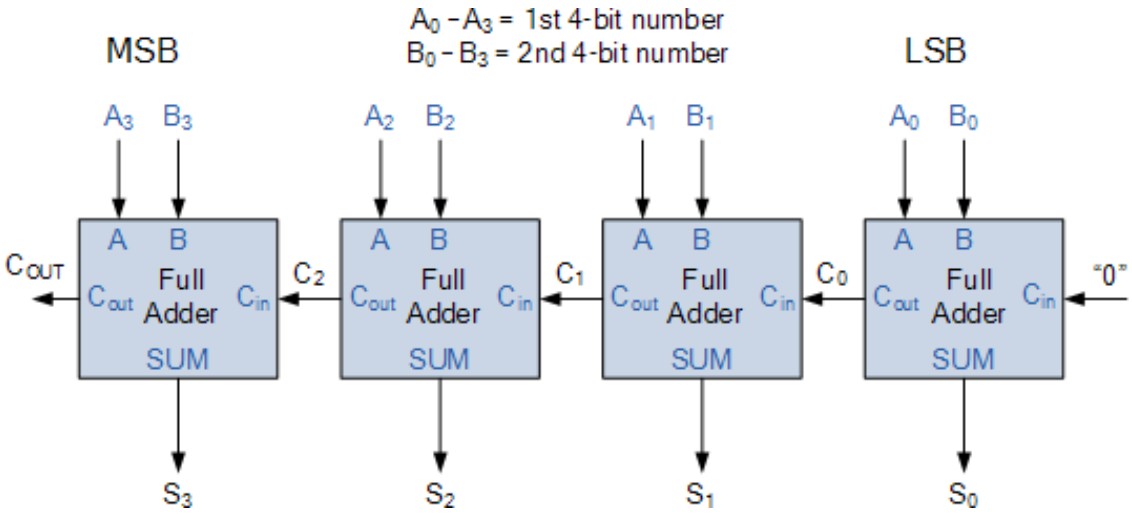
BINARY ADDER:-

- 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 C0, and it ripples through the full adders to the output carry C4. 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}

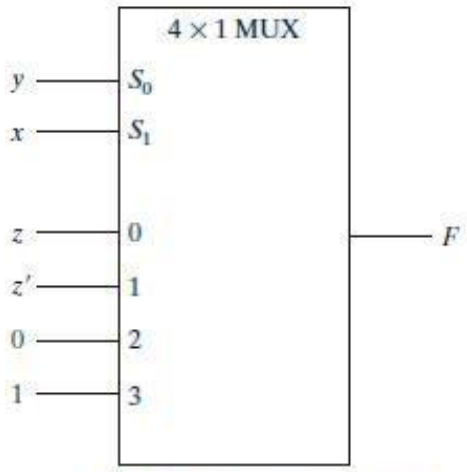
- The bits are added with full adders, starting from the least significant position (subscript 0), to form the sum bit and carry bit. The input carry C0 in the least significant position must be 0.
- The value of Ci+1 in a given significant position is the output carry of the full adder. This value is 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.



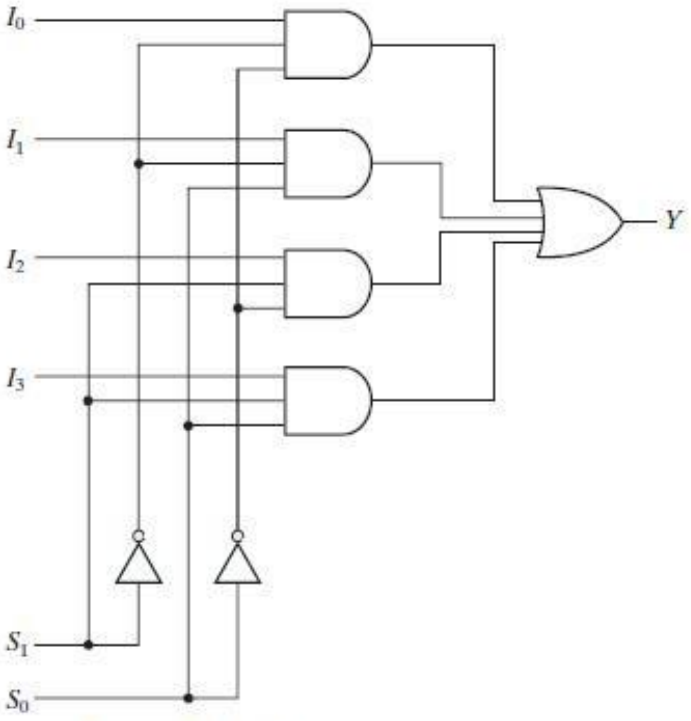
MULTIPLEXER:-

- 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 2n 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, I0 through I3, is applied to one input of an AND gate.
- Selection lines S1 and S0 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 I2 has two of its inputs equal to 1 and the third input connected to I2.
- 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 I2, 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



Logic diagram

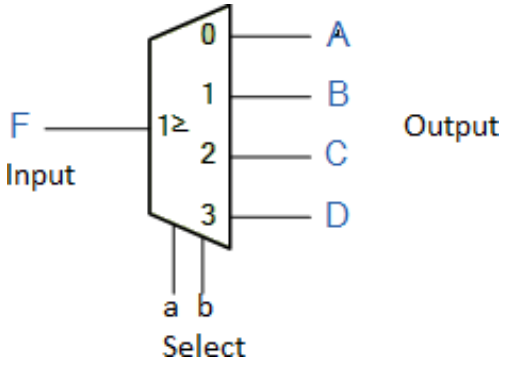
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

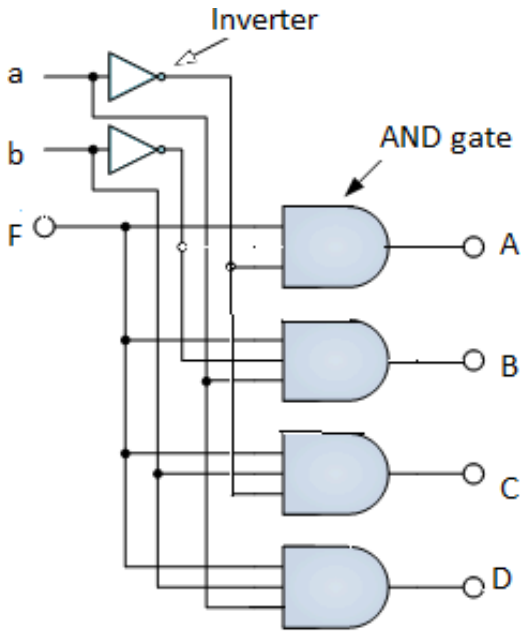
Truth table

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

ENCODER:-

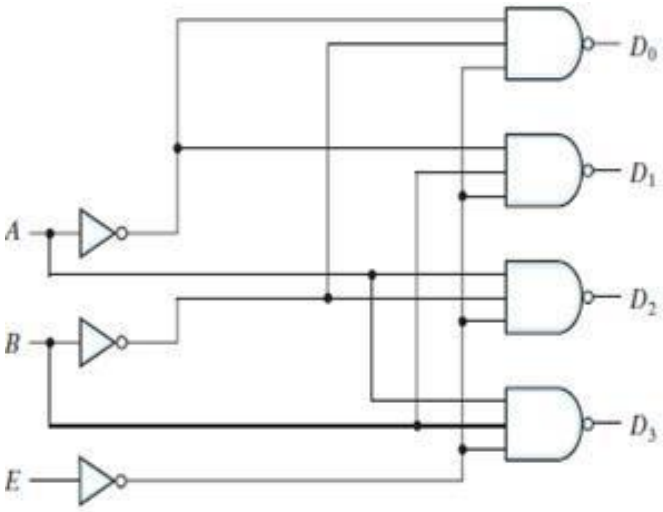
- An encoder is a digital circuit that performs the inverse operation of a decoder.
- An encoder has 2n (or fewer) input lines and n output lines.
- The output lines, as an aggregate, generate the binary code corresponding to the input value.

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 = D1 + D3 + D5 + D7
    - y = D2 + D3 + D6 + D7
    - x = D4 + D5 + D6 + D7
- 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.

DECODER:-



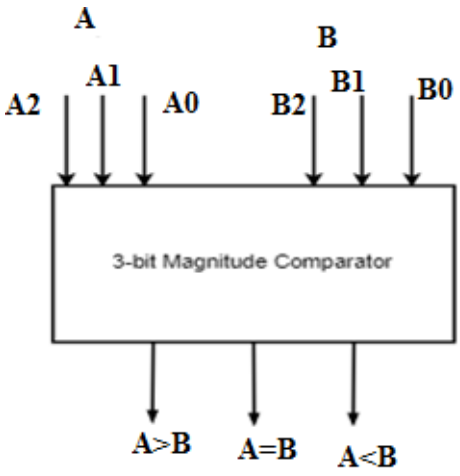
E	A	B	D_0	D_1	D_2	D_3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	1

- A decoder is a combinational circuit that converts binary information from n input lines to a maximum of 2n unique output lines.
- If the n -bit coded information has unused combinations, the decoder may have fewer than 2n outputs.
- The decoders presented here are called n -to- m -line decoders, where m … 2n.
- Their purpose is to generate the 2n (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 2n 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 D2 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.

3-bit Magnitude Comparator:-

A comparator that compares two binary numbers (each number having 3 bits) and produces three outputs based on the relative magnitudes of given binary bits is called a 3-bit magnitude comparator.



The equal functions are $A_0 = B_0$, $A_1 = B_1$, $A_2 = B_2$

Then $A=B = (A_0'B_0' + A_0B_0)(A_1'B_1' + A_1B_1)(A_2'B_2' + A_2B_2)$

The output is **A< B** in the cases of

$A_2<B_2$

$A_2 = B_2$ then $A_1<B_1$

$A_2 = B_2$, $A_1 = B_1$ then $A_0<B_0$

$$A<B = A_2'B_2 + [(A_2'B_2' + A_2B_2) * A_1'B_1] + [(A_2'B_2' + A_2B_2) *[(A_1'B' + A_1B_1) * A_0'B_0]$$

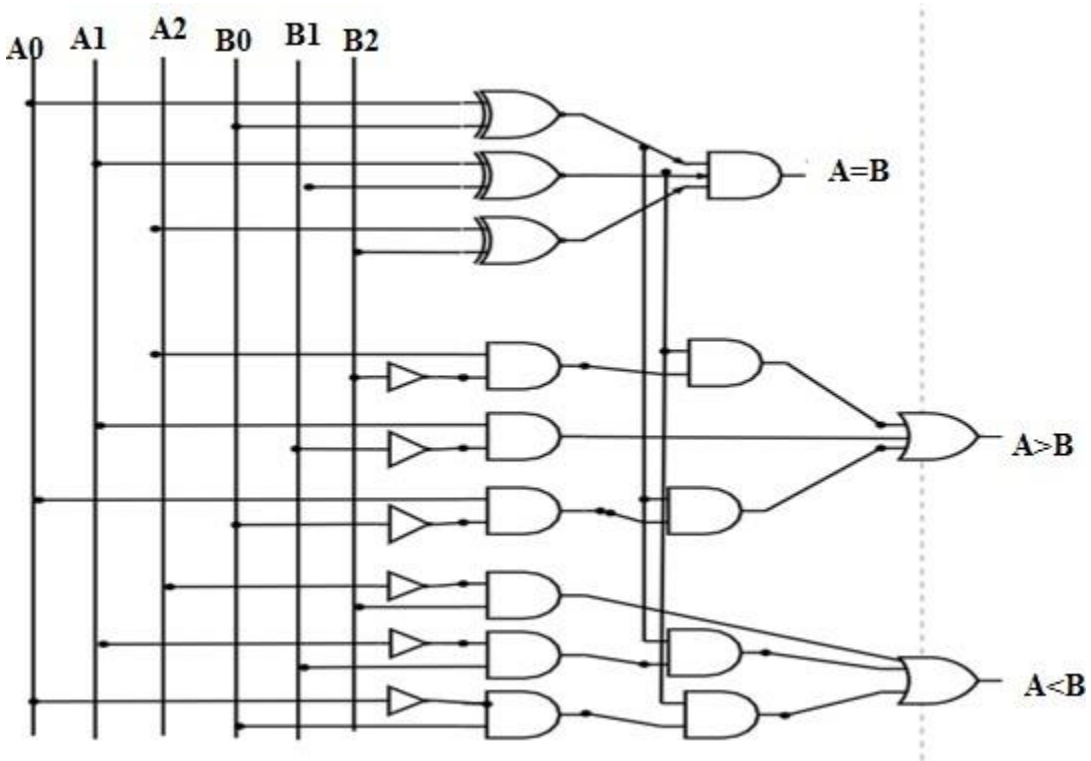
The output is **A> B i**n the cases of

$A_2>B_2$

$A_2 = B_2$ then $A_1>B$

$A_2 = B_2$, $A_1 = B_1$ **then A0>B0**

$$A>B = A_2B_2' + + [(A_2'B_2' + A_2B_2) * A_1B_1'] + + [(A_2'B_2' + A_2B_2) * [(A_1'B' + A_1B_1) * A_0B_0']$$



3-bit-logic-diagram

Applications Comparator

Digital comparator and magnitude comparator is used in different applications where data comparison is mostly required in many of the activities, and these hold many benefits too.

- Now, look into few of the applications of comparators
- Used for authorization purposes (such as password management) and biometric applications.
- These are implemented in process controllers and also in servo motor controls.
- Implemented for the data comparison of variables like temperature, the pressure is compared with that of reference values.
- Used to address decoding circuitry in computers.

Design of BCD to 7 Segment Display Decoder Circuit

The designing of BCD to seven segment display decoder circuit mainly involves four steps namely analysis, truth table design, K-map and designing a combinational logic circuit using logic gates.

The first step of this circuit design is an analysis of the common cathode seven segment display. This display can be constructed with seven LEDs in the form of H. A truth table of this circuit can be designed by the inputs combinations for every decimal digit. For instance, decimal number '1' would control a blend of b & c.

The second step is the truth table design by listing the display input signals-7, equivalent four-digit binary numbers as well as decimal number.

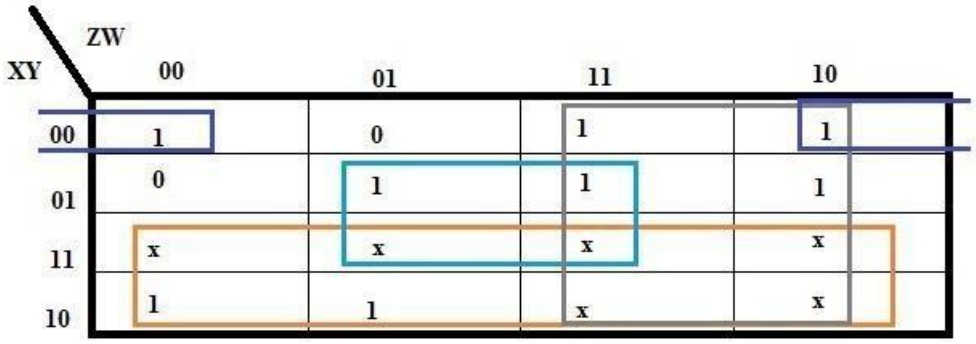
The designing of the truth table for the decoder mainly depends on the kind of display. Already we have discussed above that is, for a common cathode display, the decoder output must be high in order to blink the segment.

The tabular form of a BCD to 7-segment decoder with a common cathode display is shown below. The truth table consists of seven o/p columns equivalent to each of the seven segments. For example, the column for a-segment illustrates the various arrangements for which it is to be light up. Thus 'a'- segment is energetic for the digits like 0, 2, 3, 5, 6, 7, 8 & 9.

Digit	X	Y	Z	W	a	b	c	d	e	f	g
0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	1	1	0	0	1	1	1	1
2	0	0	1	0	0	0	1	0	0	1	0
3	0	0	1	1	0	0	0	0	1	1	0
4	0	1	0	0	1	0	0	1	1	0	0
5	0	1	0	1	0	1	0	0	1	0	0
6	0	1	1	0	0	1	0	0	0	0	0
7	0	1	1	1	0	0	0	1	1	1	1
8	1	0	0	0	0	0	0	0	0	0	0
9	1	0	0	1	0	0	0	0	1	0	0

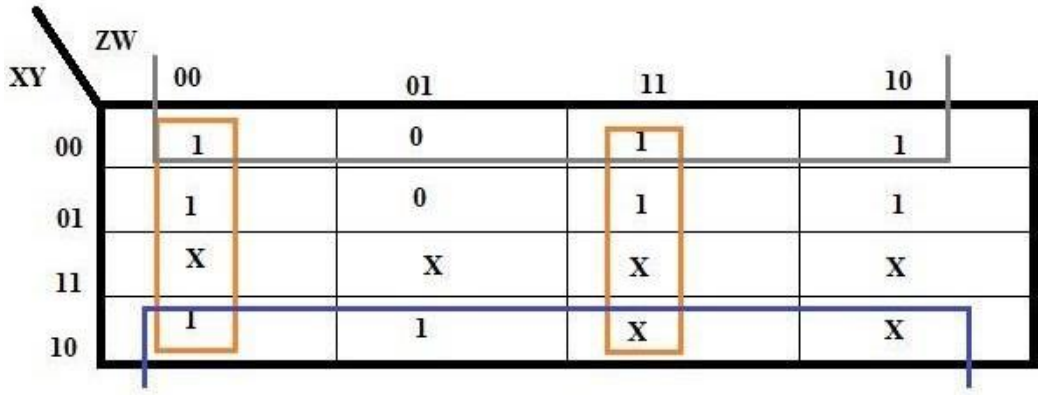
By using the above truth table, for every output function, the Boolean expression can be written.

a = F1 (X, Y, Z, W) = ∑m (0, 2, 3, 5, 7, 8, 9)



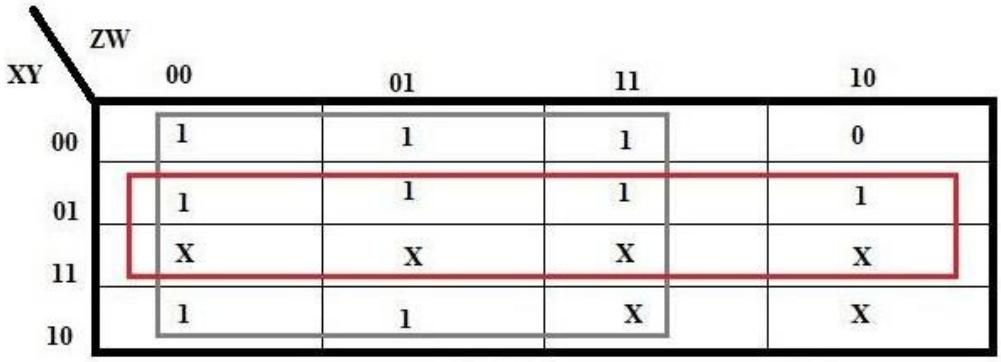
a = X + Z+ YW + YW'

b = F2 (X, Y, Z, W) = ∑m (0, 1, 2, 3, 4, 7, 8, 9)



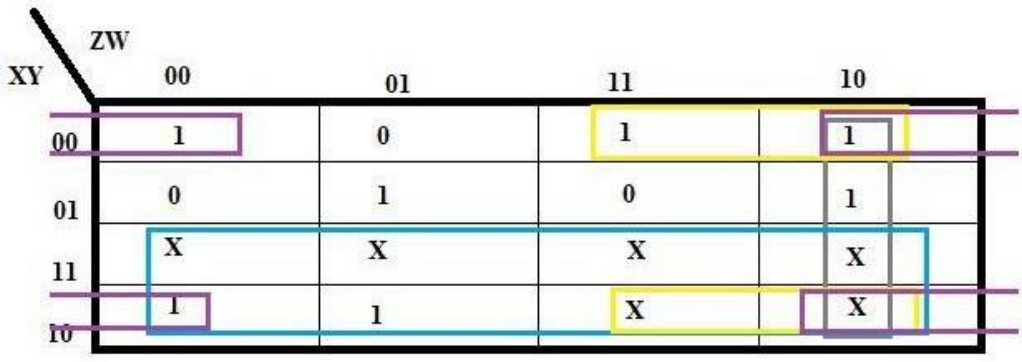
b = Y'+Z'W'+ZW

c = F3 (X, Y, Z, W) = ∑m (0, 1, 3, 4, 5, 6, 7, 8, 9)



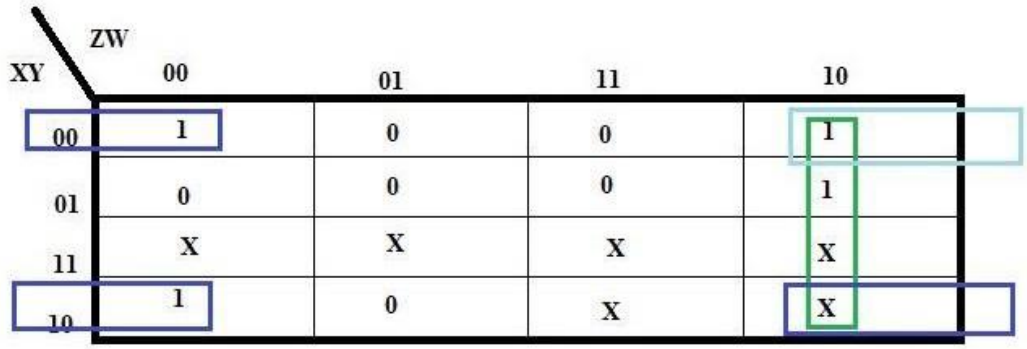
c = Y+Z'+W

d = F4 (X, Y, Z, W) = ∑m (0, 2, 3, 5, 6, 8)



d = Y'W'+ZW'+YZ'W+Y'Z +X

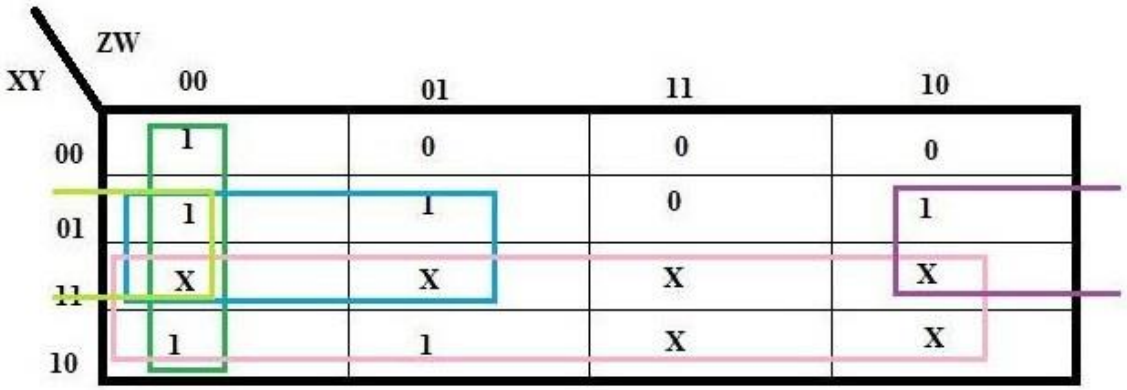
e = F5 (X, Y, Z, W) = ∑m (0, 2, 6, 8)



XY \ ZW	00	01	11	10
00	1	0	0	1
01	0	0	0	1
11	X	X	X	X
10	1	0	X	X

e = Y'W'+ZW'

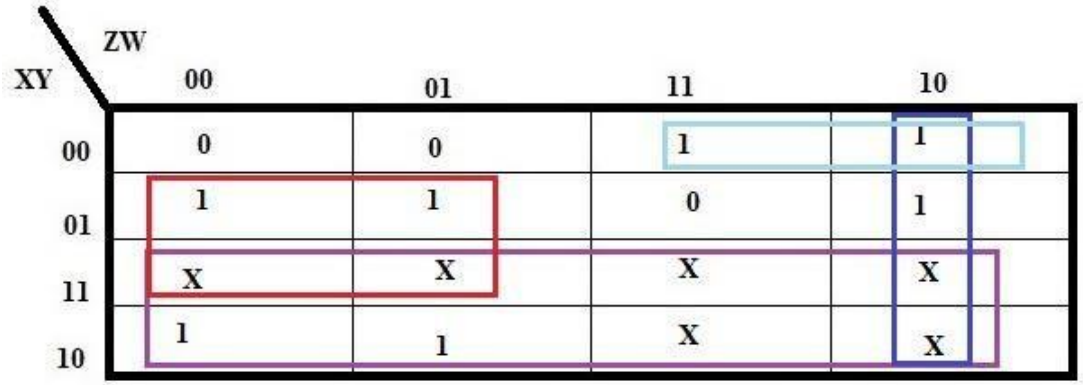
f = F6 (X, Y, Z, W) = ∑m (0, 4, 5, 6, 8, 9)



XY \ ZW	00	01	11	10
00	1	0	0	0
01	1	1	0	1
11	X	X	X	X
10	1	1	X	X

f = X + Z'W'+YZ'+YW'

g = F7 (X, Y, Z, W) = ∑m (2, 3, 4, 5, 6, 8, 9)



XY \ ZW	00	01	11	10
00	0	0	1	1
01	1	1	0	1
11	X	X	X	X
10	1	1	X	X

g = Y'Z+ZW'+YZ'+YZ'+X

The third step in this design mainly involves designing the K-map (Karnaugh's map) for every output expression as well as then shortening them to get inputs logic combination for every output.

Simplification of Karnaugh -Map

The simplification of k-map of the common cathode 7 segment decoder can be done in order to plan the combinational circuit. From the above K-map simplification, we can get the output equations like these

a = X+Z+YW+Y'W'

b = Y'+Z'W'+ZW

c= Y+Z'+W

d = Y'W'+ZW'+YZ'W+Y'Z+X

e= Y'W'+ZW'

f= X + Z'W'+YZ'+YW'

g = X+YZ'+Y'Z+ZW'

The final step of this is a designing of a logic circuit using the above k-map equations. A combinational circuit can be built by using 4-inputs namely A, B, C, D and outputs on display like a, b, c, d, e, f, g. The operation of the above logic circuit can be understood with the help of truth table only. Once all the i/ps are connected to small logic.

BCD to Seven Segment Decoder Circuit

BCD to Seven Segment Decoder Circuit

Then the combinational logic circuit's output will drive each and every one of output LEDs apart from 'g' to transmission. Therefore the number

'0' will be exhibited. Similarly, for all another grouping of the input switches, the same process would take place.

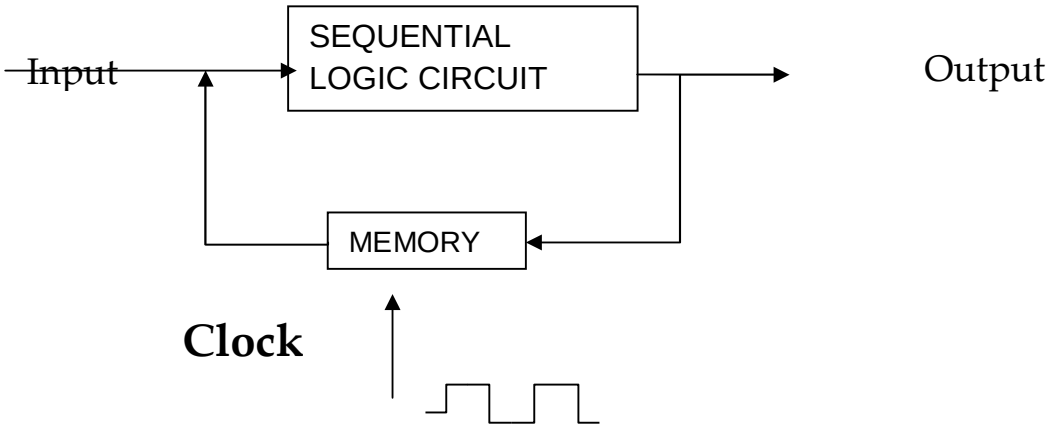
Unit-3: Sequential logic Circuits

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

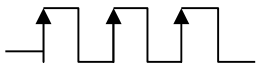
- The SLC that are controlled by clock are called synchronous SLC and those which are not controlled by a clock are asynchronous SLC.
- Clock:- A recurring pulse is called a clock.

❖ FLIP-FLOP AND LATCH:-

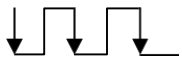
- A flip-flop or latch is a circuit that has two stable states and can be used to store information.
- A flip-flop is a binary storage device capable of storing one bit of information. In a stable state, the output of a flip-flop is either 0 or 1.
- Latch is a non-clocked flip-flop and it is the building 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.

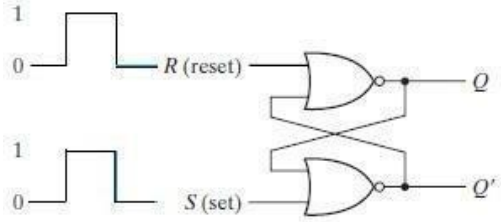


- Some common type of flip-flops include
    1. SR (set-reset) F-F
    2. D (data or delay) F-F
    3. T (toggle) F-F and
    4. JK F-F

❖ SR latch:-

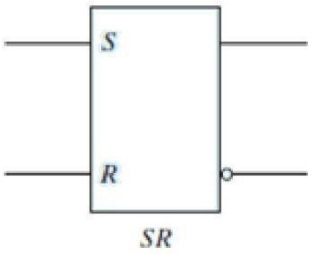
- The SR latch is a circuit with two cross-coupled NOR gates or two cross-coupled NAND gates. It has two outputs labeled Q and Q′. Two inputs are there labeled S for set and R foe reset.
- The latch has two useful states. When Q=0 and Q′=1 the condition is called reset state and when Q=1 and Q′=0 the condition is called set state.
- Normally Q and Q′ are complement of each other.
- The figure represents a SR latch with two cross-coupled NORgates.

- 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 momentary 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′	QNext	Q′Next	
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	



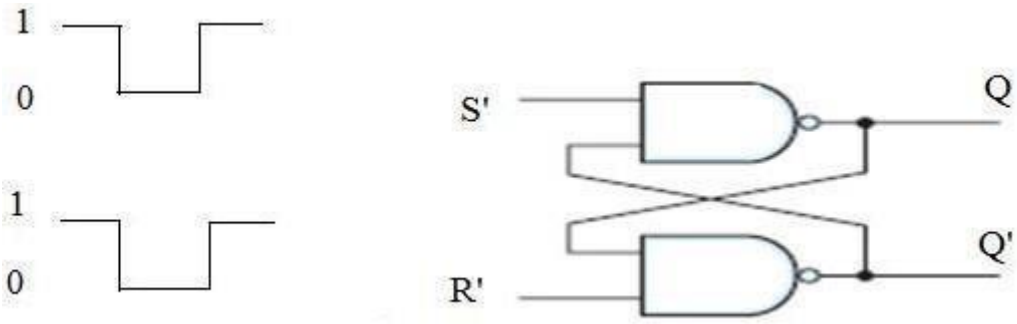
Symbol for SR NOR Latch

Racing Condition:-

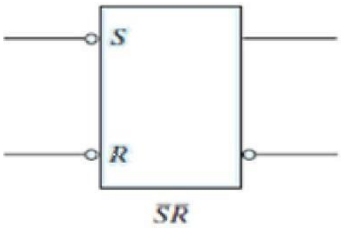
- In case of a SR latch when S=R=1 input is given both the output will try to become 0. This is called Racing condition.

- 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.

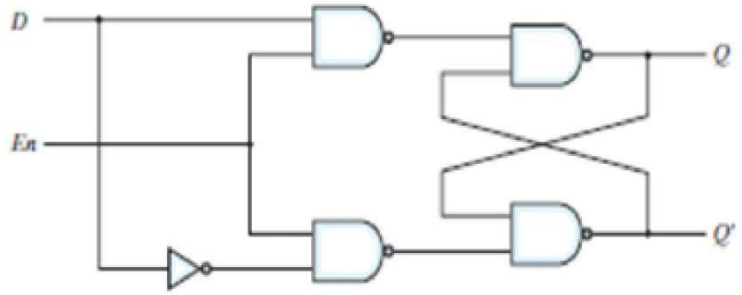


The above represents the symbol for inverted SR latch or SR latch using NAND gate. Truth table for SR latch using NAND gate or Inverted SR latch

S	R	Qnext	Q'next
0	0	Race	Race
0	1	0	1 (Reset)
1	0	1	0 (Set)
1	1	Q (No change)	Q' (No change)

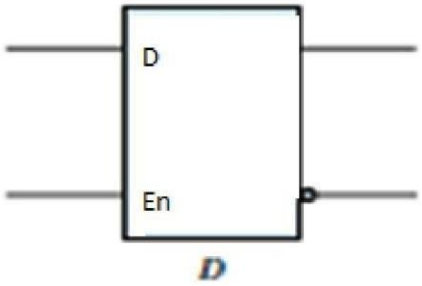
D LATCH:-

- 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)

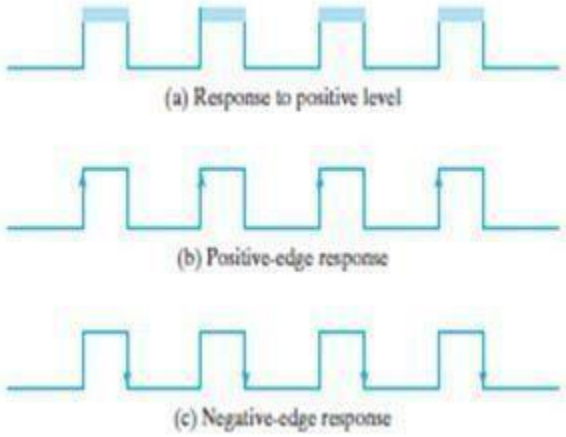
- As long as the enable input is at 0, the cross-coupled SR latch has both inputs at the 1 level and the circuit can't change state regardless of the value of D.
- The below represents the truth table for the D-latch.

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 iscalled 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.

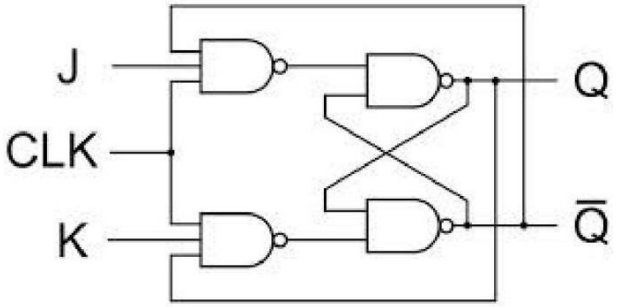


JK FLIP-FLOP:-

- The JK flip-flop can be constructed by using basic SR latch and

a clock. In this case the outputs Qand Q′ are returned back and connected to the inputs of NAND gates.

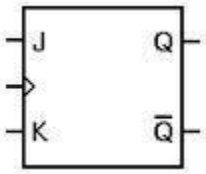
- This simple JK flip Flop is the most widely used of all the flip-flop designs and is considered to be a universal flip-flop circuit.
- The sequential operation of the JK flip flop is exactly the same as for the previous SR flip-flop with the same "Set" and "Reset" inputs.
- The difference this time 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.

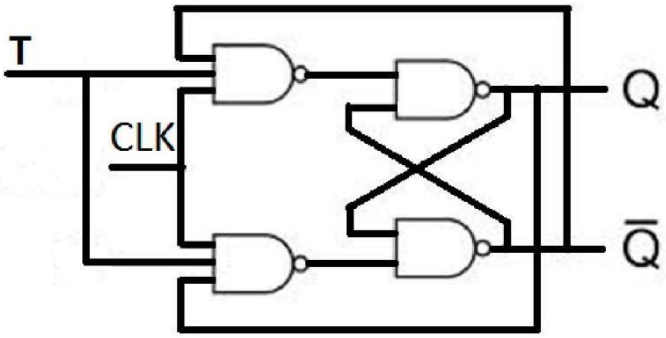
(Truth table for JK flip-flop)

Input		Output		Comment
J	K	Q	Qnext	
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.

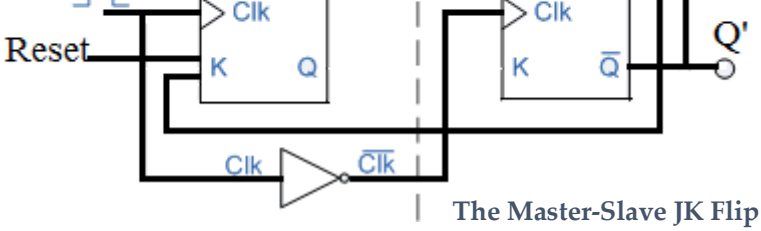


- Hence its truth table is same as that of JK flip-flop when J=K= 0 and J=K=1.So its truth table is as follow

T	Q	Qnext	Comment
0	0	0	No change
	1	1	
1	0	1	Toggles
	1	0	

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.

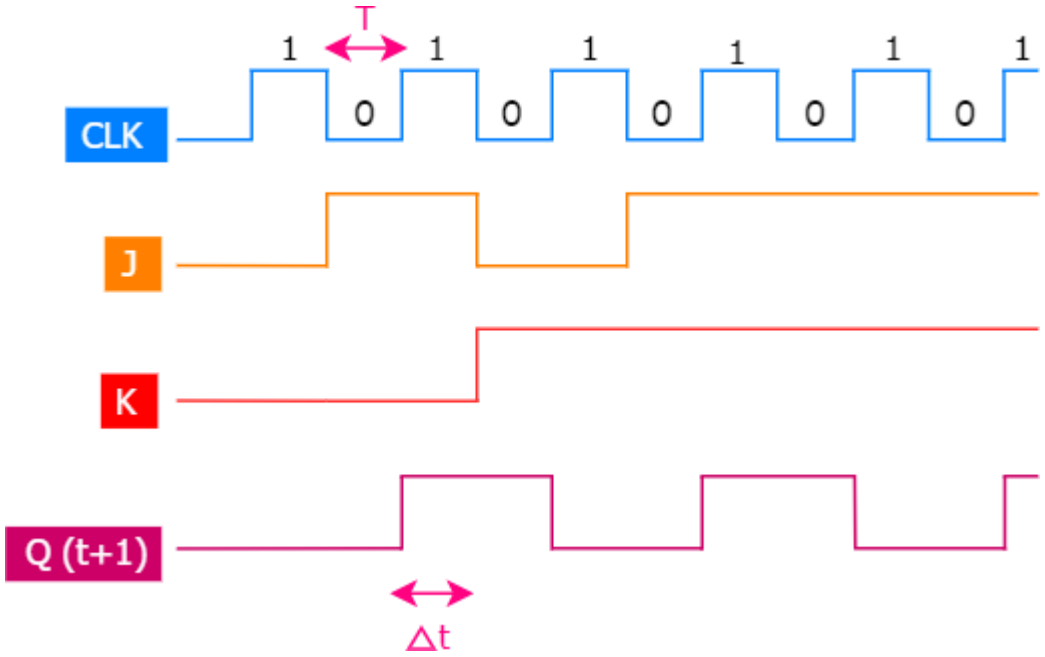
Race around condition

Introduction

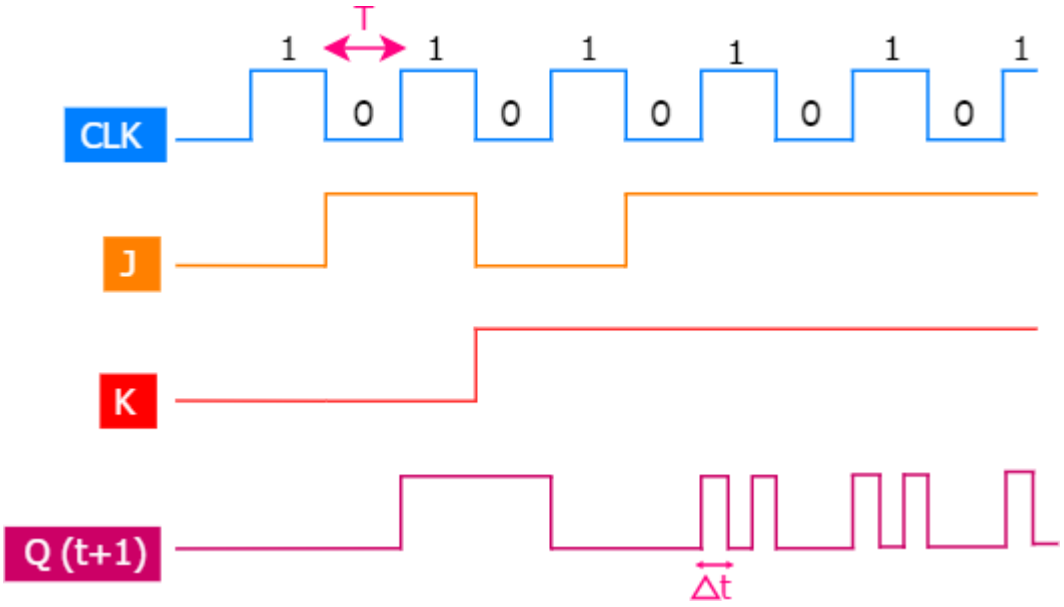
Before getting into the race around condition, let us have a look at the JK flip-flop's truth table.

Clock Input	Inputs J	K	Q	Outputs Q'	Comments		
	0	X	X	Same as previous		Same as previous	No change
	1	0	0	Same as previous		Same as previous	No change
	1	0	1	0		1	Reset
	1	1	0	1		0	Set
1	1	1	Opposite of previous	Opposite of previous	Toggle		

- Here, Q is the present state and Q' is the next state. As you can see, when J, K and Clock are equal to 1, toggling takes place, i.e. The next state will be equal to the complement of the present state.
- Now, let us look at the timing diagram of JK flip-flop



- Here, T is the time period of the clock whereas delta t is the propagation delay. The delay between input and output is called a propagation delay.
- This is what was expected, but the output may not be like this all the time. This is where **Race around condition** comes into the play.
- Let us look at the timing diagram of JK flip-flop when the race around condition is considered.



- As you already know, when J, K and Clock are equal to 1, toggling takes place. Here, propagation delay has also been reduced, so the output will be given out at the instant input is given. So there is a toggling again. Therefore, whenever Clock is equal to 1 there are consecutive toggling. This condition is called as **Race around condition**. To put it in words, " For JK flip-flop if J, K and Clock are equal to 1 the state of flip-flop keeps on toggling which leads to uncertainty in determining the output of the flip-flop. This problem is called Race around the condition. "' This condition also exists in T flip-flop since T flip- flop also has toggling options.
- **Methods to eliminate race around condition**

There are three methods to eliminate race around condition as described below:

- **Increasing the delay of flip-flop**

The propagation delay (delta t) should be made greater than the duration of the clock pulse (T). But it is not a good solution as increasing the delay will decrease the speed of the system.

- **Use of edge-triggered flip-flop**

If the clock is High for a time interval less than the propagation delay of the flip flop then racing around condition can be eliminated. This is done by using the edge-triggered flip flop rather than using the level-triggered flip-flop.

- **Use of master-slave JK flip-flop**

If the flip flop is made to toggle over one clock period then racing around condition can be eliminated. This is done by using Master-Slave JK flip-flop.

Unit-4: Registers, Memories & PLD

REGISTER:-

- 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.
- An n-bit register consists of a group of n flip- flops capable of storing n bits of binary information.
- A register consists a group of flip-flops and gates that effect their transition.
    - The flip-flops hold the binary information.
    - The gates determine how the information is transferred into the register.

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.

SHIFT REGISTER:-

- A register capable of shifting its binary information in one or both direction is called a shift register.
- All flip-flops receive common clock pulses, which activate the shift from one stage to the next.

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

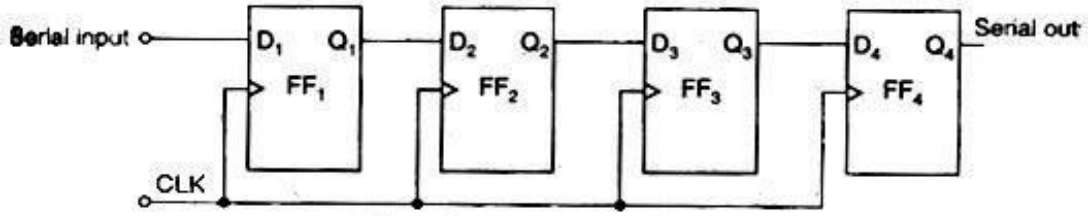
SERIAL IN, SERIAL OUT SHIFT REGISTER:-

- 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

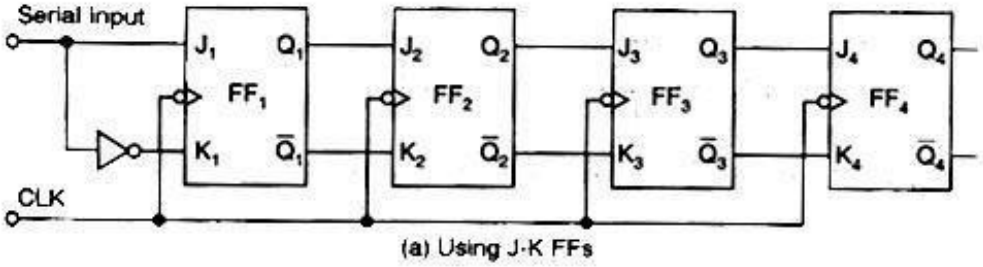
the fourth FF. The data is outputted from the Q terminal of the last FF.

- 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.

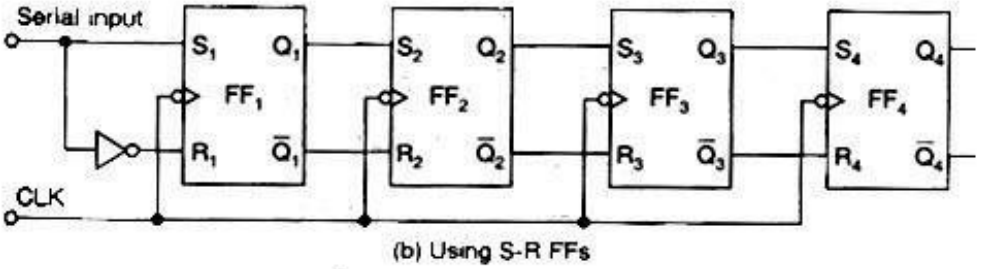
- A shift register can also be constructed using J-K FFs or S-R FFs as shown in the figure below



4-bit serial-in, serial-out, shift-right, shift register.



(a) Using J-K FFs



(b) Using S-R FFs

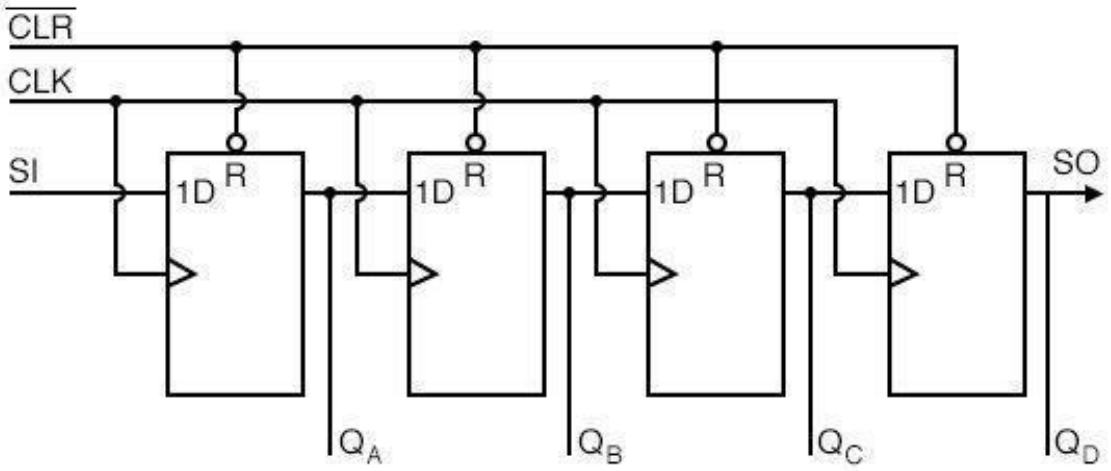
A 4-bit serial-in, serial-out, shift register.

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 serially, but the stored in the register is shifted out in the parallel form.

- When the data bits are stored once, each bits 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.



Serial-in/ Parallel-out shift register details

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

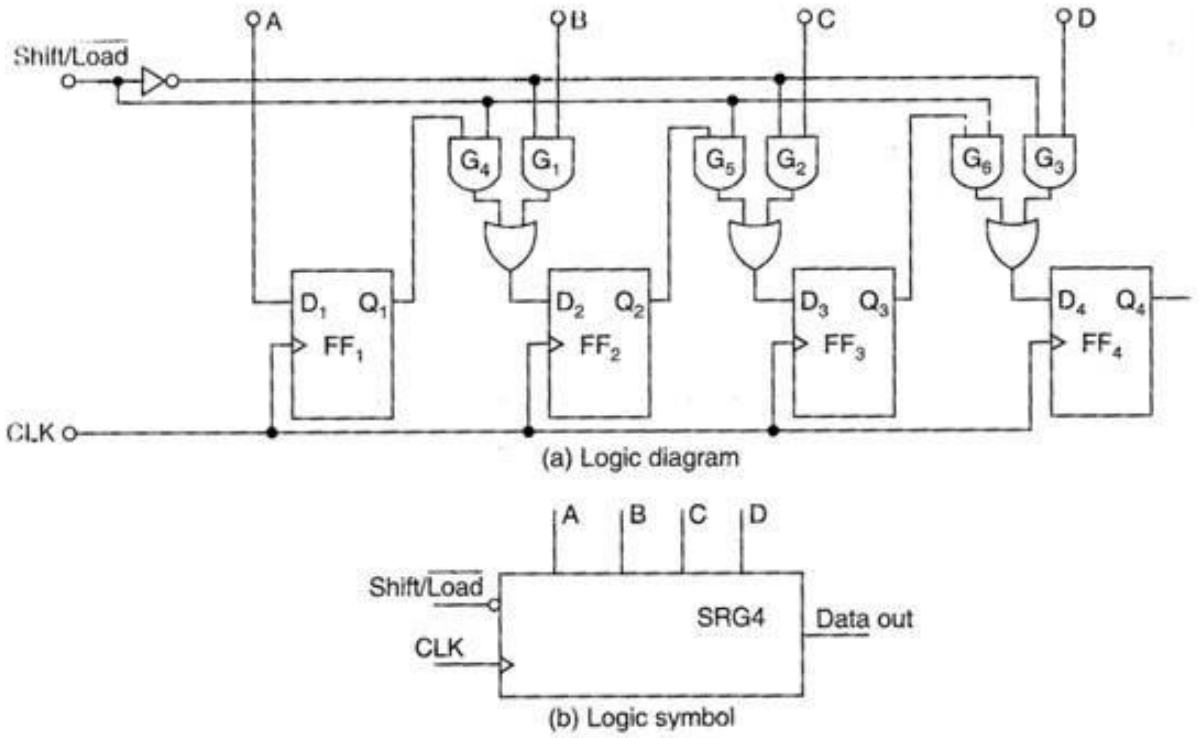
bits are transferred out of the register serially, i.e., on a bit by bit basis over a single line.

- 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 /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 Q4.

- When Shift /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 /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

/LOAD input.

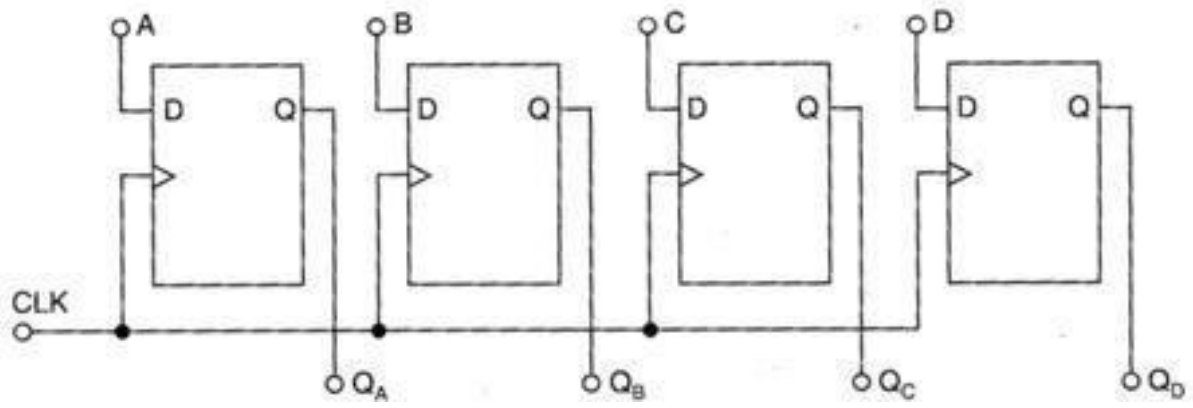


A 4- bit parallel in, serial out shift register

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

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.

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 processed with other data. A shift register can be used to delay the arrival of serial data by a subsystem where it is 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 ofdata

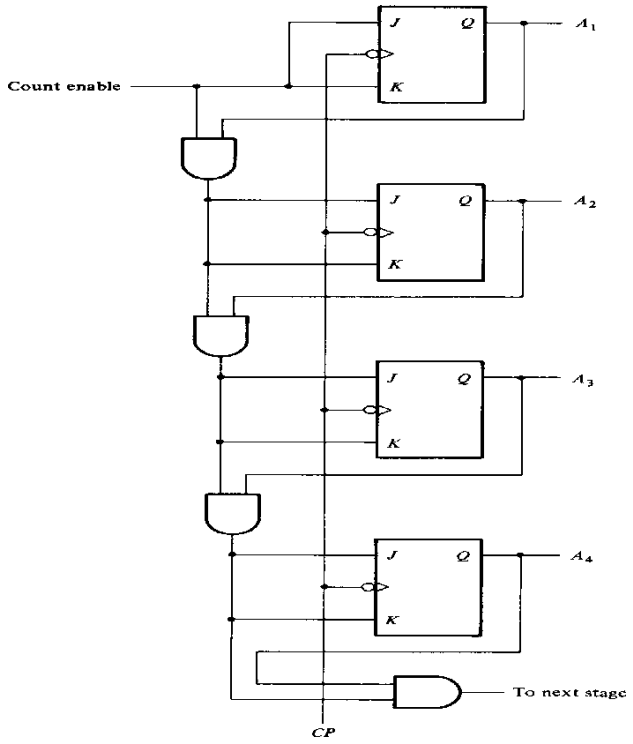
COUNTER

- 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.



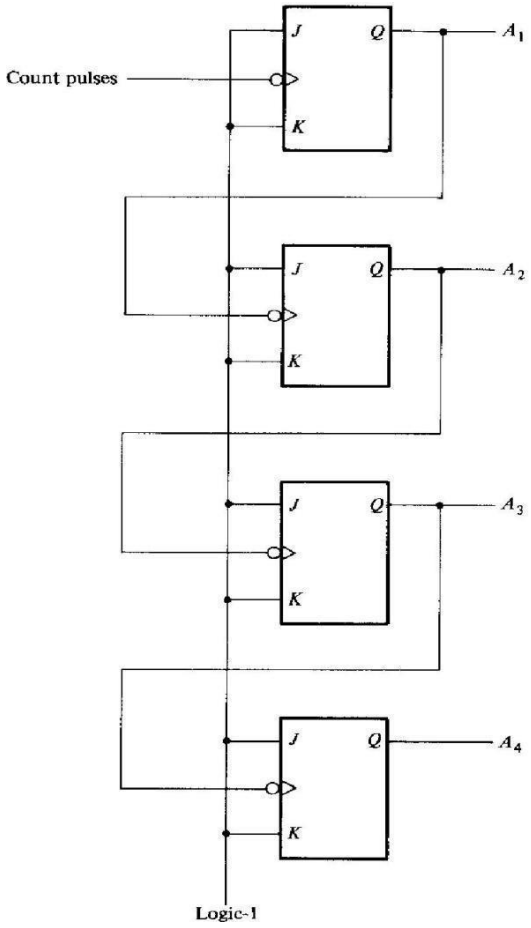
- The J and K inputs of FF0 are connected to HIGH. FF1 has its J and K inputs connected to the output of FF0, and the J and K inputs of FF2 are connected to the output of an AND gate that is fed by the outputs of FF0 and FF1.
- A simple way of implementing the logic for each bit of an ascending counter (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 2n - 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/MOD-10 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, ie. 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 QA and QD 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 QA and QD 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.

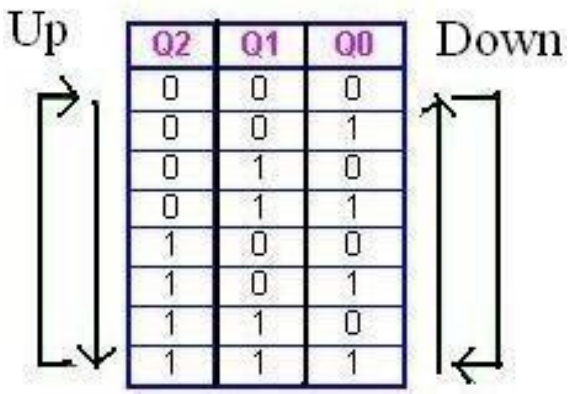
D

Decade Counter Truth Table

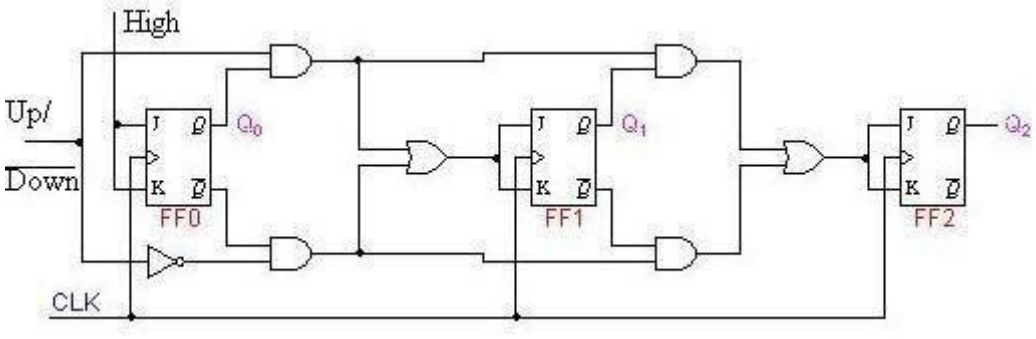
Clock Count	Output bit Pattern				Decimal Value
	QD	QC	QB	QA	
1	0	0	0	0	0
2	0	0	0	1	1
3	0	0	1	0	2
4	0	0	1	1	3
5	0	1	0	0	4
6	0	1	0	1	5
7	0	1	1	0	6
8	0	1	1	1	7
9	1	0	0	0	8
10	1	0	0	1	9
11	Counter Resets its Outputs back to Zero				

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, Q0 toggles on each clock pulse.
  - For the UP sequence, Q1 changes state on the next clock pulse when Q0=1.
  - For the DOWN sequence, Q1 changes state on the next clock pulse when Q0=0.
  - For the UP sequence, Q2 changes state on the next clock pulse when Q0=Q1=1.
  - For the DOWN sequence, Q2 changes state on the next clock pulse when Q0=Q1=0.



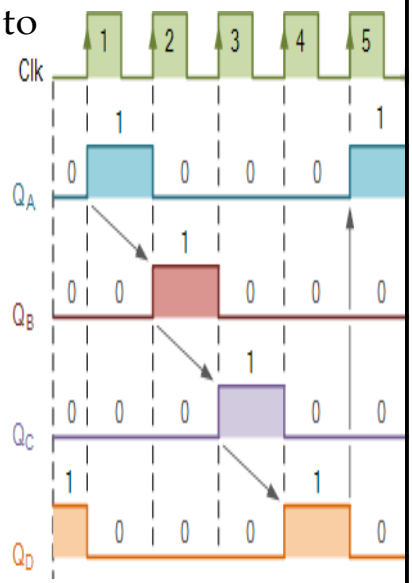
  - These characteristics are implemented with the AND, OR & NOT logic connected as shown in the logic diagram above

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.
- Then by looping the output back to the input, (feedback) we can convert a standard shift register circuit into a ring counter. Consider the circuit below.

4-bit Ring Counter:-

The synchronous Ring Counter example above, is preset so that exactly one data bit in the register is set to logic "1" with all the other bits reset to "0". To achieve this, a "CLEAR" signal is firstly applied to all the flip-flops together in order to "RESET" their outputs to a logic "0" level and hen a "PRESET" pulse is applied to the input of the first flip-flop (FFA) before the clock pulses are

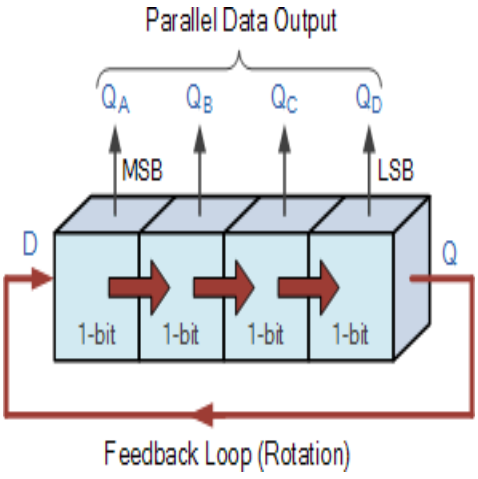


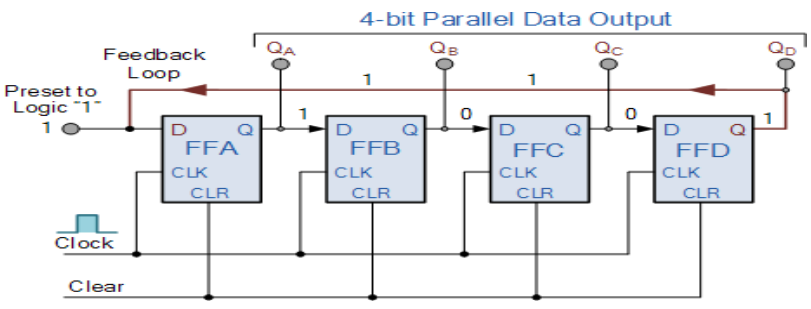
applied. This then places a single logic "1" value into the circuit of the ring counter.

So on each successive clock pulse, the counter circulates the same data bit between the four flip-flops over and over again around the ring" every fourth clock cycle. But in order to cycle the data correctly around the counter we must first "load" the counter with a suitable data pattern as all logic "0's" or all logic "1's" outputted at each clock cycle would make the ring counter invalid.

This type of data movement is called "rotation", and like the previous shift register, the effect of the movement of the data bit from left to right through a ring counter can be presented graphically as follows along with its timing diagram:

Rotational Movement of a Ring Counter





Concept of Memory

- Computer memory is a generic term for all of the different types of data storage technology that a computer may use, including RAM, ROM, and flash memory.
- Some types of computer memory are designed to be very fast, meaning that the central processing unit (CPU) can access data stored there very quickly. Other types are designed to be very low cost, so that large amounts of data can be stored there economically.
- Another way that computer memory can vary is that some types are non-volatile, which means they can store data on a long term basis even when there is no power. And some types are volatile, which are often faster, but which lose all the data stored on them as soon as the power is switched off.

Types of Computer Memory: Primary and Secondary

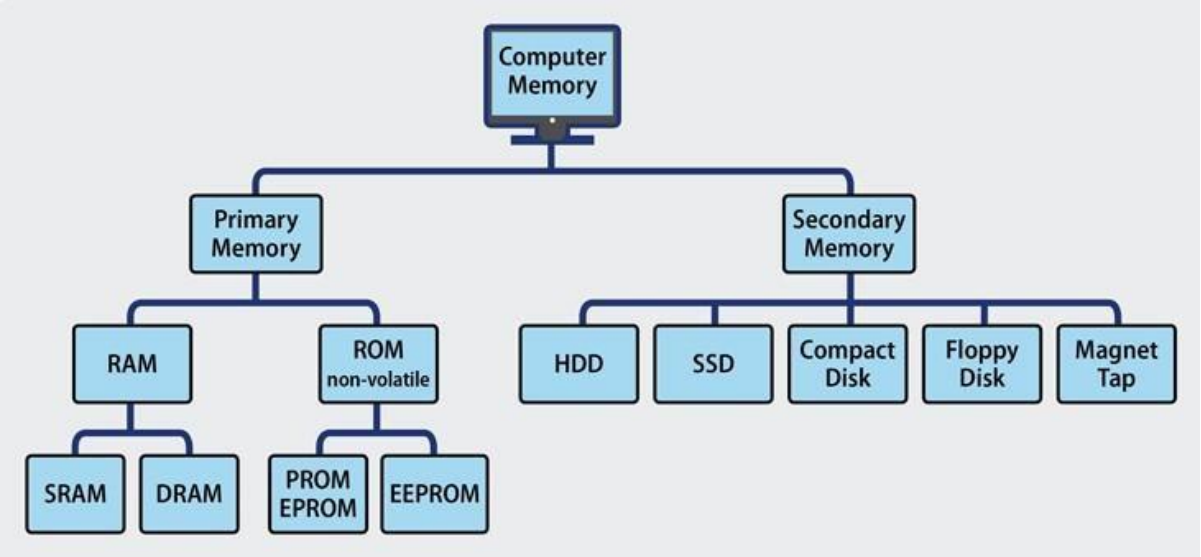
The most basic distinction is between primary memory, often called system memory, and secondary memory, which is more commonly called storage.

The key difference between primary and secondary memory is speed of access.

- Primary memory includes **ROM** and **RAM**, and is

located close to the CPU on the computer motherboard, enabling the CPU to read data from primary memory very quickly indeed. It is used to store data that the CPU needs imminently so that it does not have to wait for it to be delivered.

- Secondary memory is usually physically located within a separate storage device, such as a hard disk drive or solid state drive (SSD), which is connected to the computer system either directly or over a network. The cost per gigabyte of secondary memory is much lower, but the read and write speeds are significantly slower.



Primary Memory:-

There are two key types of primary memory:

1. RAM, or random access memory
2. ROM, or read-only memory

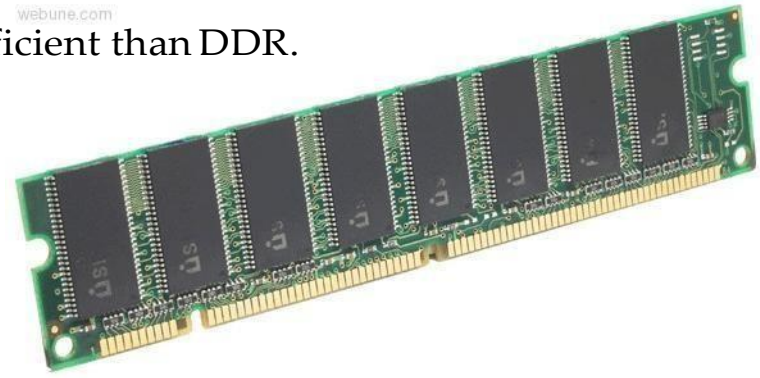
1) RAM:-

- It is also called as read write memory or the

main memory or the primary memory.

- The programs and data that the CPU requires during execution of a program are stored in this memory.
- It is a volatile memory as the data loses when the power is turned off.
- RAM is further classified into two types- SRAM (Static Random Access Memory) and DRAM (Dynamic Random Access Memory).

- **DRAM**: DRAM stands for Dynamic RAM, and it is the most common type of RAM used in computers. The oldest type is known as single data rate (SDR) DRAM, but newer computers use faster dual data rate (DDR) DRAM. DDR comes in several versions including DDR2 , DDR3, and DDR4, which offer better performance and are more energy efficient than DDR.



- **SRAM:** SRAM stands for Static RAM, and it is a particular type of RAM which is faster than DRAM,

but more expensive and bulker, having six transistors in each cell. For those reasons SRAM is generally only used as a data cache within a CPU itself or as RAM in very high-end server systems.

DRAM	SRAM
1. Constructed of tiny capacitors that leak electricity.	1.Constructed of circuits similar to D flip-flops.
2.Requires a recharge every few milliseconds to maintain its data.	2.Holds its contents as long as power is available.
3.Inexpensive.	3.Expensive.
4. Slower than SRAM.	4. Faster than DRAM.
5. Can store many bits per chip.	5. Can not store many bits per chip.
6. Uses less power.	6.Uses more power.
7.Generates less heat.	7.Generates more heat.
8. Used for main memory.	8. Used for cache.

2) ROM:-

- Stores crucial information essential to operate the system, like the program essential to boot the computer.
- It is not volatile.
- Always retains its data.
- Used in embedded systems or where the programming needs no change.
- Used in calculators and peripheral devices.
- ROM is further classified into 4 types- *ROM*, *PROM*, *EPROM*, and *EEPROM*.
- ROM is also used in simpler electronic devices to store firmware which runs as soon as the device is switchedon.

Differences between RAM and ROM

RAM	ROM
1. Temporary Storage.	1. Permanent storage.
2. Store data in MBs.	2. Store data in GBs.
3. Volatile.	3. Non-volatile.
4.Used in normal operations.	4. Used for startup process of computer.
5. Writing data is faster.	5. Writing data is slower.

PLD:-

- In a PLD (Programmable Logic Device) technology, layers implement a programmable circuit, where programming has a lower-level meaning than a

software program.

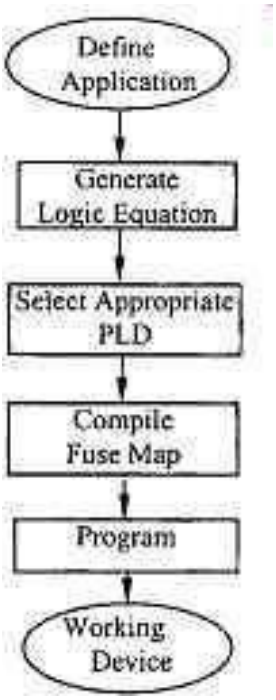
- The programming that takes place may consist of creating or destroying connections between wires that connect gates, either by blowing a fuse, or setting a bit in a programmable switch.
- Small devices, called programmers, connected to a desktop computer can typically perform such programming.
- PLD's of two types, simple and complex. One type of simple PLD is a PLA (Programmable Logic Array), which consists of a programmable array of AND gates and a programmable array of OR gates.
- Another type is a PAL (Programmable Array Logic), which uses just one programmable array to reduce the number of expensive programmable components.
- One type of complex PLD, growing very rapidly in popularity over the past decade, is the FPGA (Field

  Programmable Gate Array), which offers more general connectivity among blocks of logic, rather than just arrays of logic as with PLAs and PALs, and are thus able to implement far more complex designs. PLDs offer very low NRE cost and almost instant IC availability.

They are typically bigger than ASICs, may have higher unit cost, may consume more power, and may be slower (especially FPGAs). They still provide reasonable

performance, though, so are especially well suited to rapid prototyping.

Flowchart of PLD:-



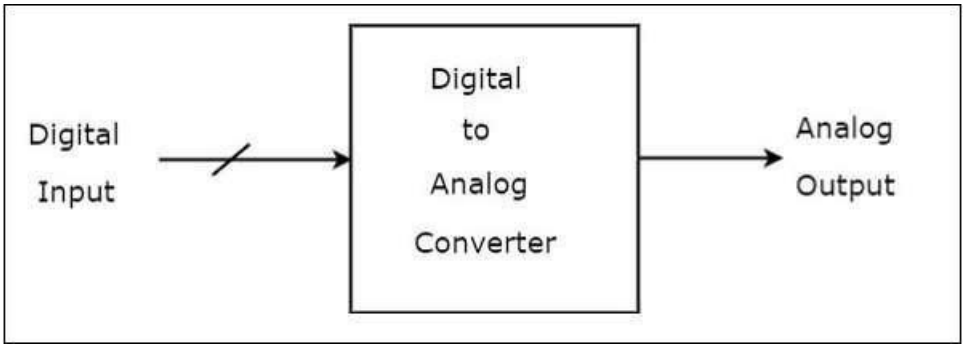
Application of PLD:-

- High performance computing
- Network Processing
- Big Data processing
- Genomics
- High Frequency trading

Unit-5: A/D and D/A Converters

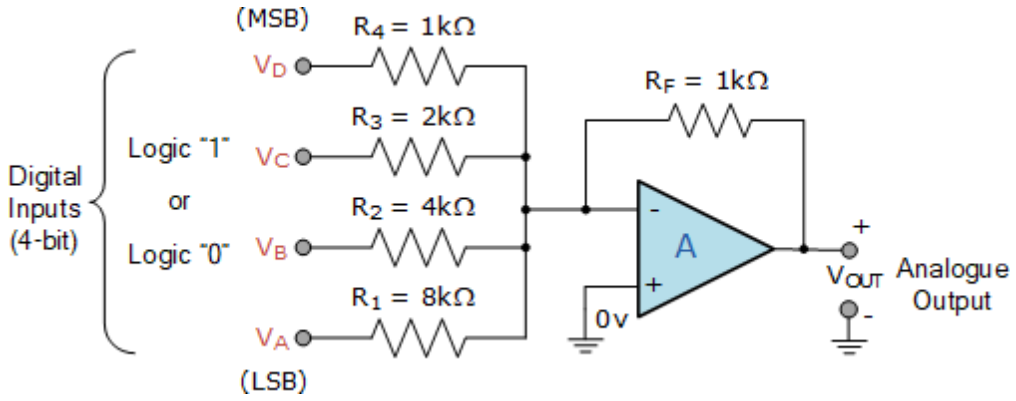
- **D/A and A/D Converter**
  - A **Digital to Analog Converter (DAC)** converts a digital input signal into an analog output signal. The digital signal is represented with a binary code, which is a combination of bits 0 and 1.

The **block diagram** of DAC is shown in the following figure −



A Digital to Analog Converter (DAC) consists of a number of binary inputs and a single output. In general, the **number of binary inputs** of a DAC will be a power of two.

- **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.



$$V_{OUT} = -\left[\frac{R_F}{R_4}V_D + \frac{R_F}{R_3}V_C + \frac{R_F}{R_2}V_B + \frac{R_F}{R_1}V_A\right]$$

$$V_{OUT} = -\left[\frac{1k\Omega}{1k\Omega}V_D + \frac{1k\Omega}{2k\Omega}V_C + \frac{1k\Omega}{4k\Omega}V_B + \frac{1k\Omega}{8k\Omega}V_A\right]$$

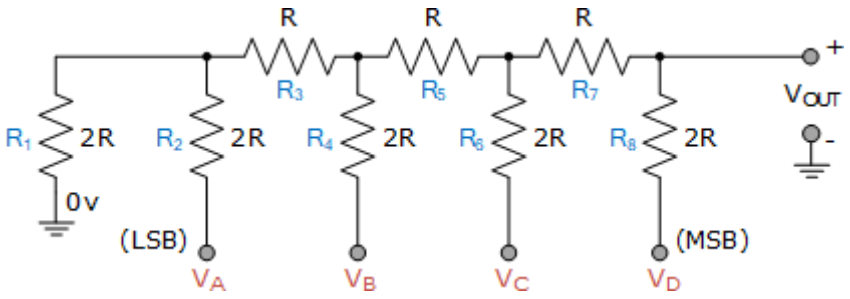
$$V_{OUT} = -\left[1V_D + \frac{1}{2}V_C + \frac{1}{4}V_B + \frac{1}{8}V_A\right]$$

Digital Inputs				V_{OUT}
D	**C**	**B**	**A**	**V/15**
0	0	0	0	0v
0	0	0	1	0.066v
0	0	1	0	0.133v
0	0	1	1	0.2v
0	1	0	0	0.26v
0	1	0	1	0.33v
0	1	1	0	0.4v
0	1	1	1	0.46v
1	0	0	0	0.53v
1	0	0	1	0.6v
1	0	1	0	0.66v
1	0	1	1	0.73v
1	1	0	0	0.8v
1	1	0	1	0.86v
1	1	1	0	0.93v
1	1	1	1	1v

PREPARED BY **-BHABODEEPIKA MOHANTY**

❖ Binary Ladder Network

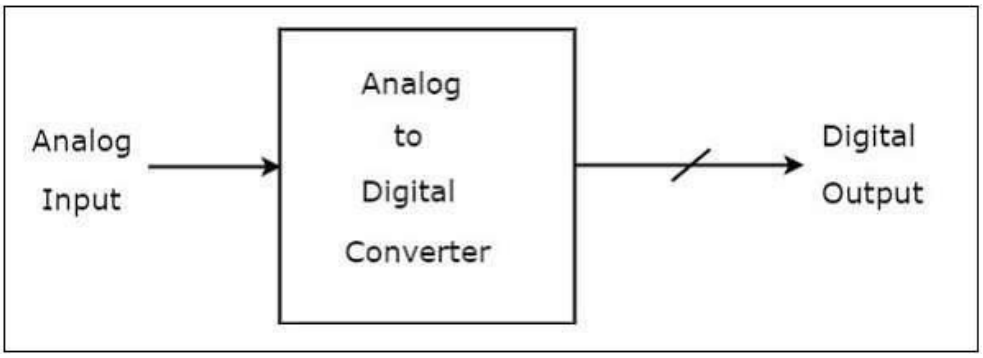
- The weighted resistor network requires a range of resistor values.
- The binary ladder network requires only two resistance values.
- From node 1, the resistance to the digital source is 2R and resistance to ground is also 2R.
- From node 2, the resistance to digital source is 2R and resistance to ground = $R + \{\frac{(2R)(2R)}{(2R+2R)}\} = 2R$ 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.



❖ Analog to Digital Converter:-

An Analog to Digital Converter **(ADC)** converts an analog signal into a digital signal. The digital signal is represented with a binary code, which is a combination of bits 0 and 1.

The **block diagram** of an ADC is shown in the following figure −



Observe that in the figure shown above, an Analog to Digital Converter **(ADC)** consists of a single analog input and many binary outputs. In general, the number of binary outputs of ADC will be a power of two.

There are **two types** of ADCs: Direct type ADCs and Indirect type ADC.

If the ADC performs the analog to digital conversion directly by utilizing the internally generated equivalent digital (binary) code for comparing with the analog input, then it is called as **Direct type ADC**.

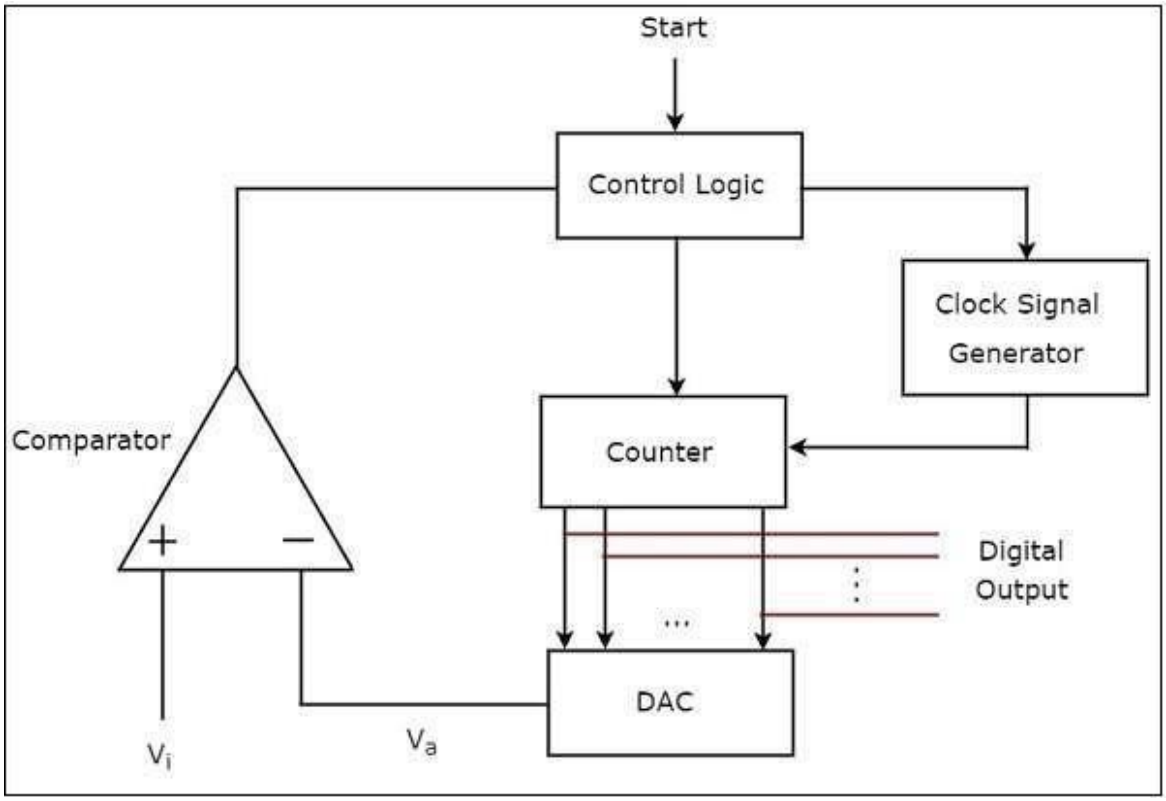
The following are the **examples** of Direct type ADCs −

- Counter type ADC
- Successive Approximation ADC

❖ Counter type ADC

A **counter type ADC** produces a digital output, which is approximately equal to the analog input by using counter operation internally.

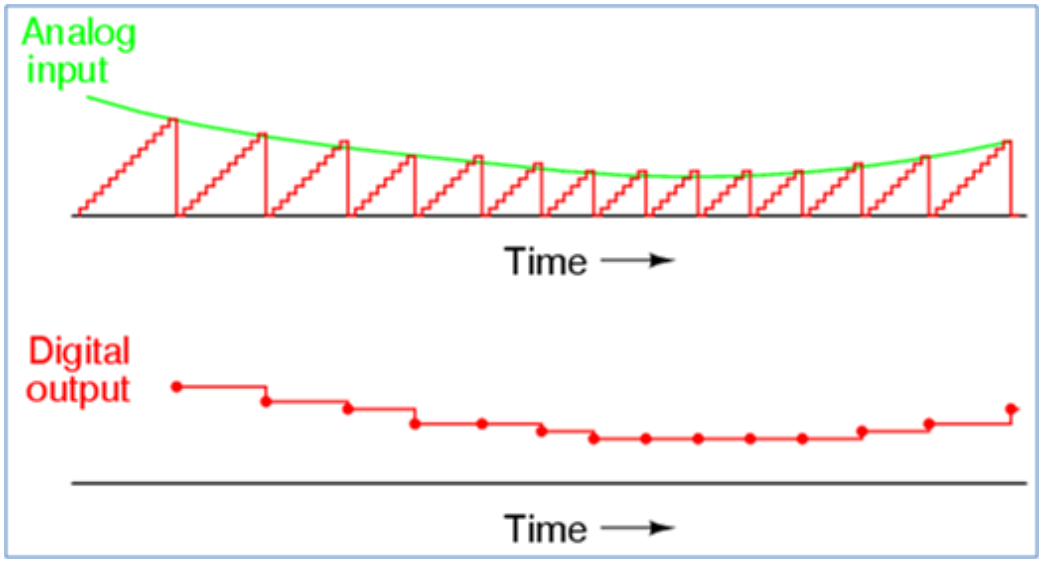
The **block diagram** of a counter type ADC is shown in the following figure −



The counter type ADC mainly consists of 5 blocks: Clock signal generator, Counter, DAC, Comparator and Control logic.

The **working** of a counter type ADC is as follows −

- The **control logic** resets the counter and enables the clock signal generator in order to send the clock pulses to the counter, when it received the start commanding signal.
- The **counter** gets incremented by one for every clock pulse and its value will be in binary (digital) format. This output of the counter is applied as an input of DAC.
- **DAC** converts the received binary (digital) input, which is the output of counter, into an analog output. Comparator compares this analog value,VaVa with the external analog input value ViVi.
- The **output of comparator** will be **'1'** as long as Vi is greater than. The operations mentioned in above two steps will be continued as long as the control logic receives '1' from the output of comparator.
- The **output of comparator** will be **'0'** when ViVi is less than or equal to VaVa. So, the control logic receives '0' from the output of comparator. Then, the control logic disables the clock signal generator so that it doesn't send any clock pulse to the counter.
- At this instant, the output of the counter will be displayed as the **digital output**. It is almost equivalent to the corresponding external analog input value ViVi.



Successive Approximation ADC

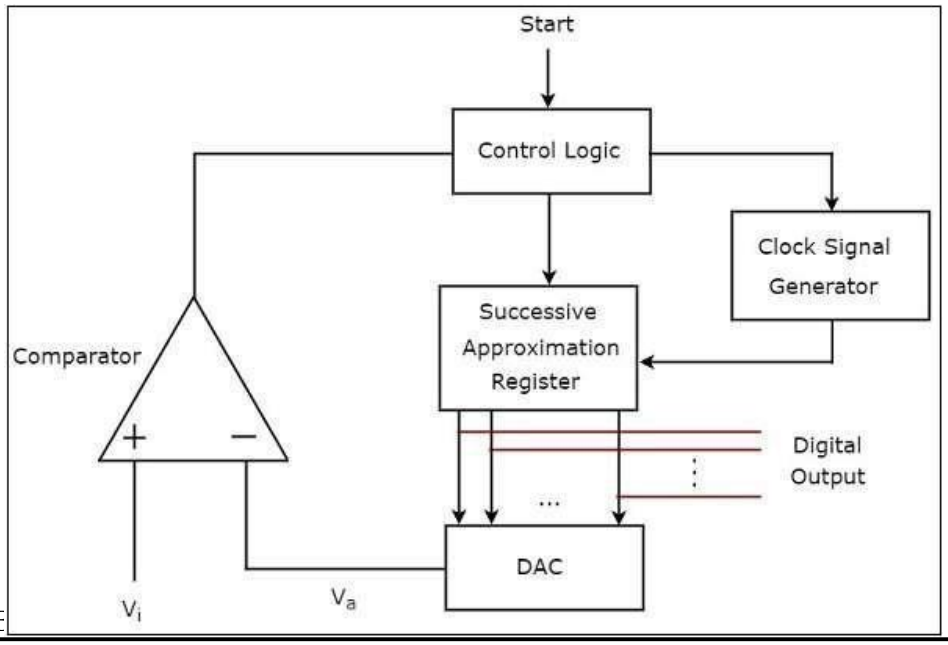
A **successive approximation type ADC** produces a digital output, which is approximately equal to the analog input by using successive approximation technique internally.

The **block diagram** of a successive approximation ADC is shown in the following figure

The successive approximation ADC mainly consists of 5 blocks− Clock signal generator, Successive Approximation Register (SAR), DAC, comparator and Control logic.

The **working** of a successive approximation ADC is as follows −

- The **control logic** resets all the bits of SAR and enables the clock signal generator in order to send the clock pulses to SAR, when it received the start commanding signal.
- The binary (digital) data present in **SAR** will be updated for every clock pulse based on the output of comparator. The output of SAR is applied as an input of DAC.
- **DAC** converts the received digital input, which is the output of SAR, into an analog output. The comparator compares this analog value VaVa with the external analog input value ViVi.
- The **output of a comparator** will be '1' as long as ViVi is greater than VaVa. Similarly, the output of comparator will be '0', when ViVi is less than or equal to VaVa.



- The operations mentioned in above steps will be continued until the digital output is a valid one.

The digital output will be a valid one, when it is almost equivalent to the corresponding external analog input value Vi.

Unit-6: LOGIC FAMILIES

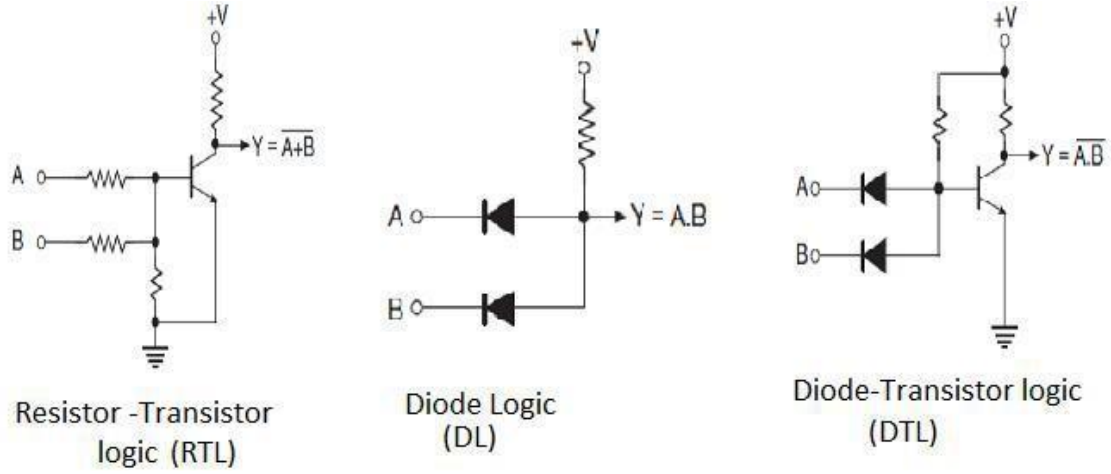
LOGIC FAMILIES

- A circuit configuration or approach used to produce a type of digital integrated circuit is called LogicFamily.
- 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, fanout, 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)
- Integrated Injection logic (I2L)

- The Bi-MOS logic family uses both bipolar and MOS devices.



Some example of DL, RTL and DTL

MOS families include:-

- ThePMOS family (using P-channel MOSFETs)
- The NMOS family (using N-channel MOSFETs)
- The CMOS family (using both N- and P-channel devices)

Characteristics of Digital IC:-

- Propagation delay
- Fan out
- Fan in
- Power dissipation
- Noise Margin
- Power Supply Requriment
- Speed of response
- Voltage level
- Current Level
- Operating Temperature
- Noise immunity

Propagation Delay time:-

- When a signal passes (propagates) through a logic circuit, it always experiences a time delay. 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:-

- 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.

Fan In:-

- It is the maximum number of inputs which the logic circuit can handle.

Power Dissipation:-

- A logic gate draws ICCH current from the supply when the gate is in the HIGH output state, draws ICCL current from the supply in the LOW output state.
- Average power isPD = VCC ICC where ICC = (ICCH + ICCL) / 2

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 :

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

Power Supply Requirement:-

- It is the maximum value of voltage and power required for operation.
- I_{HL} – High level input current. It is the minimum input current which must be supplied to the Ic to get level ”1” Voltage.
- I_{IL} – Low level input current. It the minimum input current which must be supplied to the Ic to get Level ”0” voltage.
- I_{OH}- High level output current. It the maximum output current for level “1” voltage.
- I_{OL} - Low level output current. It the maximum output current for level “0” voltage.
- V_{IH} – High level input voltage.
- V_{IL} – Low level input voltage.

- V_{OH} – High level output voltage.
- V_{IL} – Low level output Voltage.

Speed of operation:-

- The input and output pulses cannot change level simultaneously. Moreover output does not appear immediately after providing input. It means Ic takes some time to respond to the input. This is expressed as speed of operator of the Ic.

24/11/2020

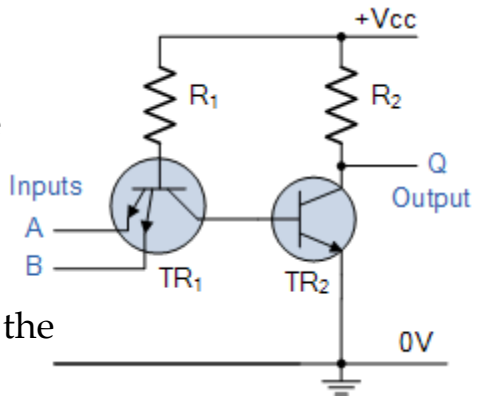
TRANSISTOR-TRANSISTOR LOGIC:-

- 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)
    - Schhottky 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

Fundamentals of TTL:-

TTL inputs are the emitter of bipolar junction transistor. In case of NAND inputs the inputs are the emitter of the multiple emitter transitor. When both the inputs are 1 the base emitter junction of multiple input transistor is reverse bias a small collector current is drawn by each of the inputs. This current passes through



Two input TTL NAND with a simple output stage

26/11/2020